École Centrale de Nantes  Université de Nantes  École des Mines de Nantes

ÉCOLE DOCTORALE STIM
« SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX »

Année 2007

---

# Placement de caméra en environnements virtuels

---

**THÈSE**

pour obtenir le grade de

**DOCTEUR DE L'UNIVERSITÉ DE NANTES**

Discipline : INFORMATIQUE

*présentée et soutenue publiquement par*

**Jean-Marie NORMAND**

*le 29 janvier 2008*

*au LINA*

devant le jury ci-dessous

| | | | |
|---|---|---|---|
| Président | : | Pr. PrénomPrésident NomPrésident | Institution |
| Rapporteurs | : | Michel Rueher, Professeur | Université de Nice-Sophia Antipolis |
| | | Pere-Pau Vázquez, Professeur | Universitat Polytechnica de Catalunya |
| Examinateurs | : | Frédéric Benhamou, Professeur | Université de Nantes |
| | | Marc Christie, Chargé de Recherche | INRIA Rennes-Bretagne Atlantique |
| | | Stéphane Donikian, Directeur de Recherche | INRIA Rennes-Bretagne Atlantique |
| | | Patrick Olivier, Senior Lecturer | University of Newcastle upon Tyne |

Directeur de thèse : Pr. Frédéric Benhamou
Encadrants de thèse : Marc Christie
Laboratoire : LINA (Laboratoire d'Informatique de Nantes Atlantique)

N° ED 0366-XXX

# PLACEMENT DE CAMÉRA EN ENVIRONNEMENTS VIRTUELS

---

*Virtual Camera Control*

**Jean-Marie NORMAND**

*favet neptunus eunti*

---

**Université de Nantes**

Jean-Marie NORMAND

***Placement de caméra en environnements virtuels***

viii+228 p.

Ce document a été préparé avec LaTeX2$_\varepsilon$ et la classe `these-IRIN` version 0.92 de l'association de jeunes chercheurs en informatique LOGIN, Université de Nantes. La classe `these-IRIN` est disponible à l'adresse :

`http://login.irin.sciences.univ-nantes.fr/`

*Impression : these.tex – 3/12/2007 – 9:41*

*Révision pour la classe : $Id: these-IRIN.cls,v 1.3 2000/11/19 18:30:42 fred Exp*

# Sommaire

# CHAPITRE 1

# Introduction

La perception visuelle regroupe l'ensemble des phénomènes optiques, chimiques et nerveux impliqués dans la discrimination par l'œil humain de la forme, de la taille, de la couleur, de la luminosité et de leurs variations au cours du temps. Jacques Aumont [Aum90], Professeur des Universités en cinéma et audiovisuel précise que l'étude de la perception visuelle débute au XIX$^{e}$ siècle par les travaux de Fechner [Fec60] et von Helmholtz [vH67], puis a évolué pour être aujourd'hui un domaine scientifique étudié par les laboratoires de psychophysique. La perception visuelle permet de comprendre les phénomènes physiologiques mis en jeu dans la transformation de l'image perçue par l'œil en influx nerveux transmis au cerveau.

Toutefois, cette notion n'est que le premier pas dans la compréhension de la relation complexe liant les images aux humains, en particulier « du pouvoir des images de nous contenir et de nous transporter »[Tis96]. Serge Tisseron, psychiatre et psychanalyste, a étudié l'évolution des dispositifs créés par les Hommes pour produire des images, depuis l'utilisation des mains, puis des crayons, pinceaux, appareils photographiques, caméras, jusqu'aux ordinateurs. Selon lui, le désir de créer des images provient du fait que l'Homme constitue le premier dispositif capable de créer ses propres images (les images psychiques : rêves, imagination, métaphores, représentations mentales, etc.). Ceci est à mettre en parallèle avec la capacité de produire des images de plus en plus « réelles », qui invitent le spectateur à rester non plus « devant » les images, mais à « entrer » dans ces dernières. Le désir de créer des mondes virtuels, interactifs, dans lesquels nous pouvons rencontrer d'autres humains par le biais d'avatars virtuels correspond à l'aboutissement de la volonté de création d'images par l'Homme. La puissance contenante de l'image laisse la place à la puissance de transformation des images.

Au contraire des psychanalystes, les sémiologues tendent à restreindre l'image aux signes qu'elle exprime. La sémiologie (ou sémiotique) est fondée sur le concept de signe, formé par la relation entre un élément perceptible, le signifiant (*e.g.* une image) et le concept associé : le signifié [WIW]. Toutefois, la théorie de sémiologie des images, ou sémiologie visuelle montre des limites. En effet, la sémiologie, instrument pensé par Peirce [Pei78] et Saussure [dS16], concerne uniquement les relations internes entre le signifiant de l'image et son référent, illustrées par le triangle sémiotique de la figure 1.1, en oubliant le problème principal posé par toute image concernant la relation que son spectateur noue avec elle (S. Tisseron [Tis96], page 129). L'image ne doit plus être pensée comme un ensemble de signes mais comme un ensemble de relations. En effet, l'image a d'abord eu une valeur de « signe » pour le développement de la pensée en Occident, en lui permettant de s'appuyer sur des images qu'elle était sans cesse appelée à dépasser. L'image a ainsi assuré à la civilisation Occidentale la maîtrise de la pensée symbolique, tout d'abord théologique et religieuse, puis scientifique et technique dans une forme de représentation symbolique du monde environnant, puis des mondes imaginaires et virtuels.

La création d'images virtuelles (*i.e.*3D), s'avère donc être la suite logique de l'évolution des dispositifs de production d'images inventés par l'Homme. L'Informatique correspond au nouvel outil mis à disposition des artistes pour la création d'images virtuelles, des metteurs en scène pour la réalisation de

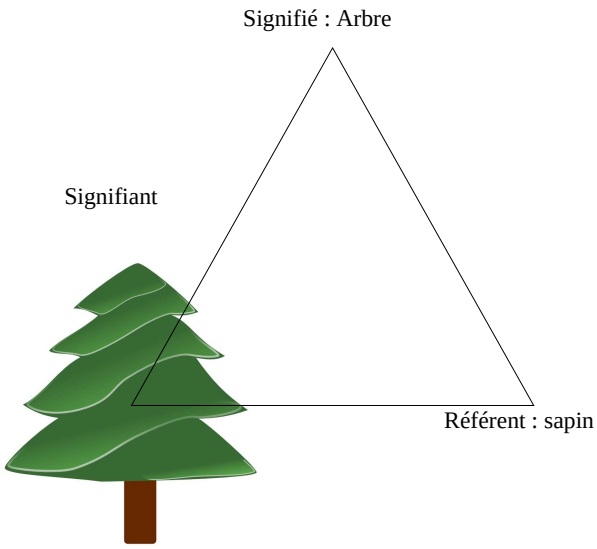



Figure 1.1 – Triangle sémiotique formé par l'image représentant un arbre (le signifiant), le concept d'arbre associé à la représentation mentale collective (le signifié) et l'espèce d'arbre représentée : un sapin (le référent).

films d'animation numériques et du public pour l'immersion virtuelle proposée par les différents moyens d'expression virtuels (jeux vidéo, films, images, etc.).

L'informatique graphique s'intéresse à la modélisation, le rendu, l'animation d'objets 2D et 3D ainsi qu'à leur manipulation par des utilisateurs. Les efforts consentis en modélisation consistent à fournir des moyens expressifs, efficaces et intuitifs de haut niveau pour la représentation d'objets complexes. Ceux-ci sont basés sur des notions mathématiques de modélisation de lignes et surfaces lisses permettant la définition de modèles à la fois visuellement agréables et industriellement réalisables. En effet, les objectifs liés à la modélisation d'objets 3D sont doubles, dépendant des applications visées : une représentation complexe et réaliste d'objets 3D en particulier pour les applications d'animation (cinéma, effets spéciaux, films d'animation, etc.) ou artistiques numériques, ou bien encore les jeux vidéo; et la possibilité de fabrication industrielle inhérentes aux applications de CAO (Conception Assistée par Ordinateur) utilisées par exemple en construction aéronautique ou automobile. Ces dernières requièrent un certain nombre de contraintes liées à la fabrication industrielle des modèles, en particulier des propriétés de continuité de normales ou de continuité de courbures concernant les surfaces utilisées pour modéliser les pièces destinées à l'usinage. Enfin, l'utilisation grandissante des scanners 3D permettant d'obtenir des nuages de points relevés à partir d'objets réels nécessite le développement d'algorithmes spécifiques pour la reconstruction de surfaces continues, à partir de points 3D dépourvus d'informations topologiques.

Le rendu consiste quant à lui en l'affichage des modèles 3D. Un des buts avoués de l'informatique graphique dans ce domaine est d'atteindre en temps réel des images indiscernables d'images réelles (films, photos, etc.). Pour ce faire, de nombreux travaux mettent en œuvre des techniques de simulation « réalistes » de la lumière et des couleurs lors du rendu d'images générées par ordinateur. De même, les

algorithmes existants profitent de nombreuses améliorations en termes de performances et bénéficient des progrès issus du matériel graphique actuel. Il existe à l'heure actuelle des images générées par informatique difficilement discernables d'images réelles, même pour des spécialistes en imagerie numérique[1]. Toutefois, le temps de calcul nécessaire à leur production n'est pas encore de l'ordre de l'interactivité. Tout comme la modélisation géométrique, le rendu profite de nombreuses avancées technologiques issues d'un nombre important de travaux académiques et/ou industriels. Le rendu est sans conteste le domaine de recherche le plus actif de l'informatique graphique.

L'interaction et la manipulation des objets 3D constituent elles aussi deux autres domaines de recherche de l'informatique graphique. Les applications industrielles et commerciales nécessitent des manipulateurs spécifiques dédiés à la modélisation d'objets 3D, en particulier à la manipulation de la souris en environnement tridimensionnel. Toutefois, bien qu'un ensemble de méthodes ait été défini dans ce but, le développement de nouvelles méthodes d'interaction s'essouffle. Un consensus semble avoir été atteint sur l'adéquation des manipulateurs existants aux besoins des utilisateurs.

Cependant, un certain nombre de problématiques ne sont pas prises en compte ou tout au moins pas suffisamment développées à notre sens : c'est le cas du contrôle de caméra en environnements virtuels. En effet, pour les « images réelles » (par opposition aux images 3D virtuelles), les dispositifs tels que la peinture, l'appareil photo, ou la caméra jouent le rôle de fixateur de l'image sur un support et permettent ensuite à l'œil de capter l'information des images pour que notre cerveau les interprètent. Au même titre que les « images réelles », les images 3D ont elles aussi besoin d'un dispositif de prise de vue adéquat : la caméra virtuelle, qui joue le rôle d'intermédiaire entre la production d'images 3D et leur interprétation mentale. Le contrôle de caméra en environnements virtuels apparaît donc fondamental dans le rôle de transmission de l'information et le manque de techniques dédiées à cette problématique se révèle marquant.

En effet, positionner et orienter une caméra dans un monde 3D est une tâche difficile. L'utilisateur possède une vision très claire du résultat qu'il souhaite obtenir en termes d'image produite, toutefois le processus classique de placement de caméra consiste en une *inversion mentale* et apparaît particulièrement contre-intuitif. Connaissant le résultat souhaité, ce processus consiste à évaluer les positions et orientations de la caméra permettant d'obtenir une image correspondante, puis à effectuer un rendu de la scène afin de vérifier si l'image obtenue est conforme au résultat souhaité. Si ce n'est pas le cas, l'utilisateur modifie alors la position et/ou l'orientation de la caméra puis répète ce processus jusqu'à ce qu'il obtienne une image satisfaisante. Le placement de caméra est donc un processus généralement long et fastidieux, et les problèmes sous-jacents se voient renforcés lorsque nous nous intéressons à l'animation de caméras virtuelles. Les méthodes classiques de définition de trajectoires de caméra, et donc d'animation de caméra, sont basées sur les notions de courbes cubiques modélisant les trajectoires et d'images clés contraignant les positions de caméra à des instants donnés de l'animation. Les trajectoires entre les différents points clés (*key-frames*) sont interpolées par des courbes de Bézier, des B-splines ou encore des *NURBS*. La position et l'orientation de la caméra sont uniquement définies aux images clé, en dehors de celles-ci ces valeurs sont interpolées et l'utilisateur n'a aucune garantie sur les valeurs de ces paramètres et donc sur la qualité de l'image produite.

Dans les applications dédiées à la création de films d'animation ou d'œuvres d'art numériques, aucune aide n'est proposée aux utilisateurs pour la prise en compte de notions cinématographiques de haut niveau. Pourtant, afin de faciliter la spécification d'images numériques, il apparaît intéressant d'avoir recours au vocabulaire issu de la photographie ou de la cinématographie pour définir des propriétés comme

[1] L'éditeur de logiciel Autodesk propose sur son site Internet un test dont le but est de distinguer des images réelles d'images numériques, afin d'illustrer le niveau de réalisme atteint par les logiciels actuels. Le test « *Fake or Foto* » est disponible à l'adresse suivante : `http://www.autodesk.com/eng/etc/fakeorfoto/quiz.html`

la définition de plans de vue classiques en cinéma ou l'utilisation des règles artistiques de composition visuelle (lignes de fuite, règle des tiers, etc.). Ainsi le placement ou le contrôle de caméra en environnements virtuels souffre d'un certain nombre de limitations au niveau des primitives de manipulation offertes aux utilisateurs. Des travaux scientifiques ont été menés dans le but de faciliter l'interaction des utilisateurs avec une caméra virtuelle, en particulier en proposant des propriétés permettant de s'abstraire d'un contrôle direct des paramètres de la caméra (*i.e.* sa position et son orientation). Toutefois, de nouveaux problèmes sont apparus en marge du développement de ces propriétés de haut niveau. En effet, alors que l'utilisateur peut décrire son problème de placement de caméra en tant que liste de propriétés devant être satisfaites (*e.g.* voir un objet dans un cadre à l'intérieur de l'image, voir un personnage de profil, etc.), il lui devient très facile de spécifier un problème insoluble (on parle également de problème sur-contraint). Les méthodes actuelles, essentiellement basées sur des approches numériques d'optimisation et/ou de résolution de contraintes, ne proposent qu'une une seule solution, *i.e.* une seule trajectoire ou une position unique de caméra. Il est pourtant facile d'imaginer la description d'un problème amenant à plusieurs solutions équivalentes en termes de respect des propriétés tout en étant différentes en termes de rendu à l'image et méritant donc toutes d'être présentées à l'utilisateur en précisant les caractéristiques propres de chaque classe de solutions. Un exemple simple concerne la définition d'une propriété utilisateur souhaitant voir un objet de profil dans une scène 3D et est illustré en figure 4.3 (page 96). Il est trivial qu'une position de caméra produisant une image de profil gauche est solution du problème au même titre qu'une caméra amenant à une image de profil droit. Les deux solutions sont donc équivalentes en ce qui concerne la satisfaction de la propriété utilisateur, mais sont différentes en termes d'image produite. Il apparaît donc intéressant de proposer à l'utilisateur les deux solutions en les caractérisant par le profil mis en évidence dans l'image résultat, et ainsi au travers d'un processus interactif de navigation dans les solutions, de mieux répondre à sa description.

Enfin, un des problèmes fondamentaux du contrôle de caméra concerne les relations d'occlusions dans les scènes et images 3D. Une occlusion survient lorsque la projection d'un objet 3D à l'écran est recouverte par celle d'un autre élément de la scène. Cette relation est primordiale en ce qui concerne les caméras virtuelles. En effet, un des objectifs principaux de la gestion d'une caméra virtuelle est la présentation d'informations à l'écran. Les objets d'intérêt de la scène (personnages principaux, objet à présenter, etc.) doivent apparaître à l'écran afin d'aider l'utilisateur à mieux comprendre la scène, les relations, les événements et leur enchaînement. Toutefois, bien que l'expression d'une occlusion soit directe, sa gestion au sein d'environnements complexes et dynamiques est difficile. Un algorithme naïf de gestion de l'occlusion consiste à projeter à l'écran toutes les faces de l'objet d'intérêt, puis toutes celles des autres objets de la scène, et tester le recouvrement des faces de l'objet d'intérêt. Un certain nombre de méthodes ont été développées afin de permettre une gestion temps réel de l'occlusion basées sur des approximations des modèles ou bien sur l'utilisation des accélérations dues au matériel graphique actuel.

L'occlusion est donc une relation algorithmiquement coûteuse et pour laquelle il existe différents niveaux d'expressivité. Nous pouvons distinguer l'occlusion partielle de l'occlusion totale d'un objet et également prendre en compte une notion plus abstraite de reconnaissance d'objets à l'écran. Un objet peut ainsi ne pas être considéré comme occulté si le spectateur est capable de le reconnaître sans ambiguïté, même partiellement occulté. C'est le cas par exemple lorsqu'un acteur passe derrière les branches d'un arbre ou derrière une barrière. Nous pouvons ainsi distinguer les occlusions de type géométrique dont l'expression est basée sur le recouvrement des faces à l'image de l'occlusion cognitive, de plus haut niveau, concernant la notion de reconnaissance d'un objet à l'écran. Étonnamment, l'occlusion n'est que peu étudiée dans les travaux concernant le contrôle de caméra. Ces travaux s'intéressent exclusivement à l'occlusion géométrique et proposent donc une expressivité restreinte. En effet, rares sont les approches

qui différencient occlusion partielle d'occlusion totale.

Dans cette thèse, nous tentons de lever un certain nombre de verrous scientifiques identifiés dans les approches existantes consacrées au placement de caméra en environnements virtuels. La première limitation concerne la prise en compte des problèmes de contrôle de caméra sur-contraints. Nous proposons dans le chapitre 3 une nouvelle approche au placement de caméra en tant que problème de satisfaction de contraintes (CSP). L'utilisateur peut spécifier indépendamment un problème potentiellement insoluble ou au contraire possédant une infinité de solutions, la méthode de résolution que nous proposons permet de lui présenter les meilleures solutions en termes de satisfaction de contraintes. Si le problème est sur-contraint, nous calculons toutes les zones de l'espace de recherche maximisant le nombre de propriétés (ensemble de contraintes) satisfaites.

La prise en compte des problèmes sur-contraints n'est toutefois pas suffisante, une caractérisation précise des propriétés satisfaites par les solutions à un problème donné permet la définition d'un nouveau type d'interaction avec l'utilisateur. Afin de caractériser précisément les solutions proposées, nous avons développé la méthode des *volumes sémantiques*, présentée au chapitre 4. Cette approche permet de diviser et de caractériser l'espace de recherche en fonction de propriétés cinématographiques spécifiées sur les objets d'une scène 3D. Nous sommes ainsi en mesure de dresser pour chacune des solutions calculées la liste des propriétés cinématographiques (*e.g.* profil droit, plan américain, etc.) satisfaites en fonction de sa position 3D. Notre méthode introduit un nouveau type d'interaction avec l'utilisateur lors de la spécification et de la résolution de problèmes de placement de caméra en environnements virtuels.

Enfin, nous avons précisé que la gestion de l'occlusion nous apparaît essentielle pour le contrôle de caméra et que les travaux menés à ce sujet restent étonnamment marginaux. Nous proposons une approche de gestion de l'occlusion permettant la prise en compte d'un couple d'objets, et extensible à un nombre arbitraire d'objets d'intérêt, dans un environnement virtuel complexe et dynamique. Les travaux et résultats correspondants sont présentés dans le chapitre 5.

Cette thèse s'organise de la façon suivante : avant de présenter nos contributions à la problématique de placement de caméra en environnements virtuels, nous dressons un état de l'art des méthodes existantes. Notre première contribution concernant une approche numérique pour le placement de caméra en environnement virtuel est développée dans le chapitre 3. L'approche des volumes sémantiques est ensuite présentée dans le chapitre 4. Ensuite, notre travail concernant la gestion de l'occlusion d'objets d'intérêt en environnements temps réel dynamique est décrite en chapitre 5. Enfin, nous dressons dans le chapitre 6 un bilan et proposons des pistes de travaux futurs concernant chacune de nos contributions, avant de conclure.

Soucieux de permettre au lecteur de se familiariser avec les modèles de représentation 3D des objets et d'une caméra virtuelle, nous adoptons tout au long de ce manuscrit, sauf mention contraire, les mêmes modèles de représentation pour ces entités. Ceux-ci sont présentés en annexe A, page 219.

# CHAPITRE 2

# État de l'art

« Le cinéma est pour moi un art tridimensionnel. Avec ma caméra, j'ai le sentiment de sculpter l'espace. »
David CRONENBERG

« Si je savais prendre une bonne photographie, je la ferais chaque fois. »
Robert DOISNEAU

Nous dressons dans ce chapitre un état de l'art des techniques consacrées au placement de caméra, ainsi qu'à la planification de trajectoires de caméra en environnements virtuels. Nous réunissons dans la suite de ce chapitre ces deux notions sous le terme de contrôle de caméra. En s'inspirant des techniques utilisées en photographie et en cinématographie, ainsi que pour utiliser l'expérience acquise en termes de composition visuelle et de montage, un large panel de méthodes ont été proposées par la communauté scientifique dans le but de contrôler de façon interactive ou automatique une caméra virtuelle. En effet, le contrôle de caméra joue le rôle de dénominateur commun pour bon nombre d'applications allant de la visualisation de données scientifiques, des visites virtuelles (musées, villes, etc.), au contrôle ou à l'examen d'objets (usinés ou virtuels), des jeux vidéo et jusqu'à la notion de « conte virtuel » (*virtual storytelling*). La diversité des objectifs visés par chacune de ces applications induit néanmoins un traitement particulier à apporter à la caméra. La définition de méthodes génériques de contrôle de caméra adaptables au plus grand nombre d'applications possible apparaît donc comme un défi particulièrement relevé.

Nous identifions quatre catégories d'approches consacrées au problème de contrôle de caméra, illustrées en figure 2.1 :

- les approches interactives, proposent un ensemble de procédures permettant de mettre en correspondance (nous parlons pas la suite de « mapper » ou de « mappage » des paramètres) les paramètres de l'utilisateur obtenus par le biais de périphériques d'entrée (tels que souris, clavier, *trackball*, etc.) avec les degrés de liberté de la caméra virtuelle.
- les approches réactives gèrent la caméra comme une entité autonome, c'est-à-dire sans interaction avec l'utilisateur éventuel. Le comportement de la caméra est déterminé en fonction de propriétés identifiées dans l'image ou de tâches visuelles à accomplir, *e.g.* le suivi de cible mobile.
- les approches à base de planification de trajectoires de caméra, dont le but est de générer des trajectoires de caméra satisfaisant un ensemble de propriétés (absence de collisions, etc.) dans un environnement virtuel. Ces approches sont basées sur des approches robotiques de planification de mouvement (*motion planning*) dont l'objectif est de calculer des trajectoires pour les déplacements d'un robot.
- les approches déclaratives, quant à elles reflètent une volonté de s'affranchir d'un contrôle direct des paramètres de la caméra. Elles tendent à évoluer vers un plus haut niveau de manipulation de la caméra dans lequel l'utilisateur spécifie non plus des propriétés relatives à la caméra, mais des propriétés visuelles qu'il souhaite retrouver dans l'image ou dans la séquence d'images finale.

Chacune de ces classes d'approches peut être subdivisée en fonction des applications visées, de la nature des techniques employées et de l'expressivité offerte à l'utilisateur.

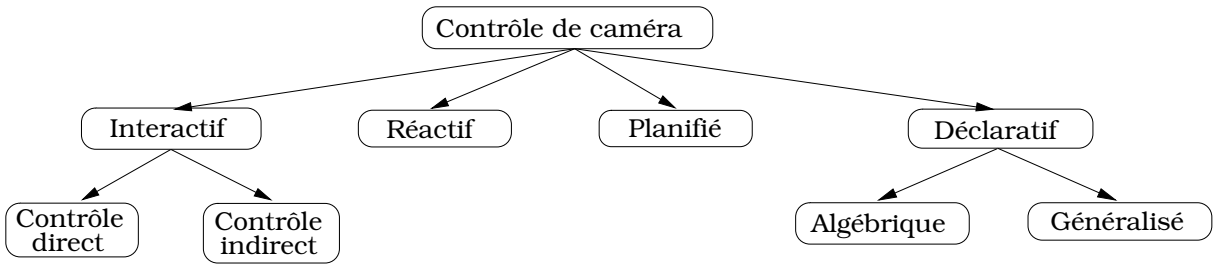



Figure 2.1 – Taxonomie des approches de contrôle de caméra.

Bien que bon nombre des approches de contrôle de caméra aient été développées en réponse à un problème particulier et possèdent donc des spécificités, elles partagent également des caractéristiques communes. Toutes doivent prendre en compte la complexité intrinsèque d'une caméra virtuelle traduite à la fois par son nombre de degrés de liberté (7), la difficulté inhérente à la notion de planification de trajectoires ainsi que l'évaluation de la notion d'occlusion, primordiale pour le contrôle de caméra.

Au long de cet état de l'art, après avoir introduit les motivations liées au contrôle de caméra dans un ensemble d'applications cibles, nous présentons l'évolution des méthodes de contrôle de caméra, présentées en figure 2.1. Nous commençons par une description des approches de manipulation interactive de la caméra, poursuivons par une section consacrée aux techniques réactives, nous détaillons ensuite les méthodes basées sur la planification de trajectoires, avant de terminer par une présentation des approches déclaratives, à savoir les approches algébriques, les techniques d'optimisation et les méthodes basées sur la satisfaction de contraintes. Enfin, nous concluons ce chapitre par une discussion sur les différences d'expressivité offertes à l'utilisateur par chacune de ces classes de méthodes.

## 2.1 Motivations

Cette section présente les différentes exigences liées au contrôle de caméra dans un ensemble d'applications clés en informatique graphique. Cette analyse nous permet de dresser une liste tant des besoins que des difficultés inhérentes à la modélisation et à la résolution de problème de contrôle de caméra.

### 2.1.1 Modeleurs 3D conventionnels

Dans les environnements classiques de modélisation 3D, une caméra est généralement définie par sa position dans la scène et par la donnée de deux vecteurs représentant respectivement son orientation (vecteur vision ou *look-at*) et une direction verticale (ou *up* vecteur). Les techniques d'animation de caméra sont basées sur des méthodes d'interpolation classiques de courbes *splines* liées à l'utilisation de points de contrôle et d'images clés (ou *keyframes*). Les *splines* sont des courbes cubiques qui offrent l'avantage de pouvoir modifier localement un des points de contrôle les définissant, sans pour autant perturber l'apparence globale de la courbe, contrairement aux courbes de Bézier, cf. chapitre sur la représentation des courbes et surfaces du livre de Foley *et al.* [FvDFH90].

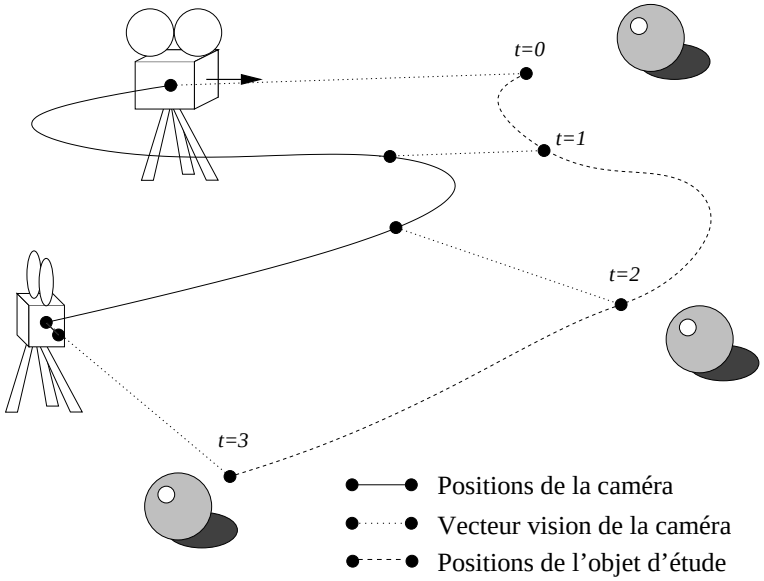



Figure 2.2 – Exemple de spécification d'une trajectoire de caméra dans un environnement classique de modélisation 3D.

La figure 2.2 illustre un exemple d'animation de caméra dans un modeleur classique. Comme nous pouvons le remarquer sur cette figure, construire une trajectoire de caméra revient à interpoler sa position, son orientation et son vecteur *up* étant donné des images clés représentant l'évolution du déplacement d'un objet suivi au cours du temps. Une maîtrise fine de la vitesse de la caméra est possible *via* la manipulation des graphes de vélocité de chacune des courbes composant la trajectoire.

Les modeleurs actuels offrent également à l'utilisateur un ensemble d'outils de plus haut niveau permettant de contraindre aisément les degrés de liberté de la caméra (en particulier le vecteur *look-at*) en fonction d'un objet unique (mobile ou immobile) dans la scène. Certains d'entre eux permettent également de spécifier des paramètres de décalage de la cible de la caméra. Cette dernière pourra, par exemple, filmer la scène un peu « en avant » ou « en arrière » de l'objet d'intérêt afin d'anticiper ses mouvements. Le contrôle de caméra peut également être facilité par la mise à disposition de métaphores physiques visant à simplifier les tâches de suivi de cible. Citons par exemple la métaphore de « barre de remorquage » virtuelle (*virtual rod*) qui permet de lier la caméra à un objet d'intérêt dans la scène et ainsi d'effectuer conjointement leurs déplacements. De par les possibilités d'extension des modeleurs actuels, il est possible d'ajouter aux contrôles basiques de caméra de nouveaux manipulateurs par le biais du développement de greffons (*plug-ins*) spécialisés ou par l'utilisation de langages de scripts. Enfin, l'utilisation de données sensorielles obtenues par expérimentations sur des caméras réelles pour la spécification de trajectoires de caméras virtuelles connaît actuellement un essor notable.

Il est à noter qu'en pratique, le modèle de représentation des trajectoires de caméra (deux courbes splines) n'est pas adapté à la reproduction numérique (*i*) ni des techniques et comportements utilisés par les caméramans, (*ii*) ni des caractéristiques et propriétés mécaniques des caméras réelles. Étonnament, et en dépit du fait de l'existence de nombreux travaux basés sur l'expression de propriétés cinématographiques pour décrire les mouvements de caméra, les modeleurs actuels n'ont pas tenté d'incorporer de

telles techniques de manière native. Même des fonctionnalités très simples à mettre en œuvre, comme la notion de cadrage (*framing*) permettant de spécifier l'appartenance d'un objet à un cadre de l'écran défini par l'utilisateur (cf. section 4.2.5) ne sont pas proposées (à l'exception d'un outil de cadrage simpliste incorporé au logiciel Softimage XSI 3D[www]). L'expressivité offerte aux utilisateurs concernant la définition de notions cinématographiques dans la scène modélisée est proche de zéro. Toutes les spécifications doivent se faire à l'aide des fonctionnalités de bas niveau en modifiant directement les courbes splines caractérisant la trajectoire de la caméra dans la scène, ou en agissant directement sur les paramètres de la caméra.

Cette inadéquation des environnements de modélisation 3D à un contrôle de haut niveau de la caméra peut s'expliquer par la généricité que souhaitent atteindre les principaux éditeurs de ces logiciels. En effet, les modeleurs ne sont pas uniquement destinés à la réalisation de films d'animation 3D, mais par exemple au domaine de la CAO (Conception Assistée par Ordinateur). Il apparaît donc plus intéressant de proposer dans les environnements 3D des fonctionnalités profitables à la majorité des domaines d'application plutôt que de se focaliser sur les techniques de contrôle de caméra de haut niveau profitables principalement aux films d'animation. De plus, bien que les idiomes cinématographiques soient généralement basés sur la représentation de personnages à l'écran (comme dans les notions de gros plan, plan américain, etc.), il est nécessaire de pouvoir les appliquer à toutes sortes d'objets. Une description sémantique (et non plus seulement géométrique) des objets est donc nécessaire. Enfin, malgré la convergence sémantique de la grammaire et des termes employés en cinématographie, la simple description d'un plan (*shot*) en terme de propriétés classiques peut amener à une multitude de solutions équivalentes. Cette équivalence des solutions en termes de rendu visuel à l'écran n'est pourtant pas prise en compte dans les solutions actuelles.

En dépit de toutes ces difficultés, l'intégration de manipulateurs de haut niveau basés sur les notions cinématographiques permettant la spécification de positions et de trajectoires de caméras représente une plus-value significative sur les modèles existants de contrôle de caméra.

### 2.1.2 Jeux vidéo

Les jeux vidéo 3D représentent la plate-forme de test (*benchmark*) par excellence pour les techniques de contrôle de caméra. Un problème classique de contrôle de caméra en environnement virtuel consiste au suivi d'un (éventuellement de plusieurs) personnage(s) dans une scène 3D dynamique et fortement peuplée (en objets, décors ou en personnages), tout en évitant les occlusions à l'écran. Ce problème correspond exactement au rôle que doit jouer le système de gestion de caméra (*camera manager*) dans un jeu vidéo. De plus, il n'est pas rare à l'heure actuelle de voir les jeux 3D incorporer des aspects narratifs à l'intrigue en ayant recours à des plans de vue judicieusement choisis à la fois pendant (*in-game*) et en dehors (*cut-scenes*) des périodes effectives de jeu. Les jeux vidéo regroupent donc dans une même application les critères essentiels d'évaluation d'un système de contrôle de caméra, à savoir un contrôle fin des paramètres de bas niveau de la caméra, ainsi qu'une prise en compte de propriétés narratives ou esthétiques de haut niveau; il est donc logique de leur porter une attention particulière lorsque nous étudions le contrôle de caméra.

De par l'augmentation régulière de la qualité visuelle des jeux 3D (nombre de polygones affichés, ou effets spéciaux pris en compte, cf. figure 2.3) et de celle conjointe du réalisme des moteurs physiques utilisés, les algorithmes temps-réel déployés pour la gestion des caméras se doivent d'être toujours plus performants. Ces derniers sont généralement basés sur des heuristiques très rapides, et donc simplistes, de vérification d'occlusion telles que le lancer de rayons (*ray-casting*). Pour plus d'informations sur la gestion de l'occlusion, le lecteur est invité à se reporter au chapitre 5.

(a) Capture d'écran du jeu Crysis développé par Crytek© (b) Capture d'écran du moteur 3D Unreal Engine ©Epic Games.

Figure 2.3 – Captures d'écran temps réel de jeux vidéo de dernière génération.

Malgré cette apparition de graphismes proches du photoréalisme (cf. illustrations 2.3) et l'utilisation de plus en plus répandue d'intrigues complexes, le contrôle de caméra dans les jeux vidéo a été largement négligé. Pourtant, John Giors [Gio04] (développeur dans un studio de jeux vidéo, Pandemic Studios) fait remarquer le rôle primordial de la caméra : « *it is the window through which the player interacts with the simulated world* ». Les jeux vidéo actuels, et plus particulièrement les jeux sur console, font montre de plus en plus d'engouement pour l'incorporation d'aspects narratifs à l'intrigue. Ils ne souhaitent plus seulement être basés sur de l'action, mais veulent offrir aux joueurs un nouveau type d'expérience plus cinématographique que par le passé. Cette volonté illustre la nécessité de transcrire les règles et conventions cinématographiques au niveau du contrôle de caméra.

Toutefois, nous devons remarquer que les jeux vidéo sont différents du cinéma en ce qui concerne le contrôle de caméra. En effet, dans les jeux vidéo, la caméra est dirigée directement ou indirectement par les joueurs. Les mouvements de caméra sont inférés d'après les mouvements des personnages contrôlés par les joueurs. De plus, les jeux représentent un environnement dynamique temps-réel dans lequel la caméra se doit de répondre immédiatement à la moindre action exercée par le joueur (ou l'environnement) sur le personnage d'intérêt.

Le développement de contrôle de caméra basé sur les principes de composition visuelle permet néanmoins la mise en exergue automatique d'objets ou d'actions importantes à effectuer, c'est-à-dire présenter de l'information au joueur, comme le fait un réalisateur pour son public. L'utilisation de techniques de montage, en particulier lors de cinématiques (*cut-scenes*) ou de sauts narratifs (*jump-cuts*), permet d'augmenter le *gameplay* en guidant le joueur vers des objectifs à atteindre.

Il faut néanmoins remarquer que l'utilisation de techniques cinématographiques, tout comme celles de montage automatique, reste l'exception plutôt que la règle dans les systèmes de contrôle de caméras vidéoludiques.

*Full Spectrum Warrior* [ww6w], un jeu d'action de simulation militaire développé par Pandemic Studios, a montré une avancée notable en ce qui concerne la gestion de la caméra. Dans ce jeu, le joueur doit contrôler une escouade de soldats. Le nombre important d'objets à garder à l'écran rend la tâche de contrôle de caméra difficile. Les développeurs ont mis en place le système « *auto-look* » qui permet de

maintenir à l'écran des images contenant le moins d'occlusions possibles de l'escouade de soldats grâce à l'utilisation d'une technique innovante de tracé de rayons (*ray-casting*). Le même procédé fût utilisé lors des scènes vues d'hélicoptère (*fly-by*) afin d'éviter les collisions avec des objets composant le décor. Lorsqu'aucune solution n'est viable, en particulier lorsque des objets empêchent le déplacement de la caméra, des sauts narratifs sont alors utilisés.

(a) Le joueur ne voit pas l'environnement dans lequel se déroule l'action, son personnage étant filmé de face. Ce plan de vue nuit à la jouabilité et à la perception de son environnement.

(b) Le joueur a une meilleure représentation de l'espace autour de lui. Il peut réagir aux événements de l'action grâce à la vue en « *over the shoulder* ».

Figure 2.4 – Illustration de deux placements de caméra du jeu à la troisième personne : « Tomb Raider: L'Ange des Ténèbres » ©Eidos interactive.

Il existe trois grandes classes de systèmes de contrôle de caméra dans les jeux vidéo :

- Première personne (*first person*) : le joueur contrôle directement la caméra, ce qui accentue le sentiment d'immersion. Une grande variété de jeux utilisent des systèmes de caméra à la première personne, ils sont regroupés sous l'appellation de FPS (pour *First Person Shooter*) pour lesquels nous citons la série des *Doom* ou des *Quake* développés par *idSoftware* [www]. La gestion de la caméra dans ces jeux ne pose aucun problème particulier puisqu'elle consiste à mapper directement les actions du joueur (limitées à des déplacements et/ou sauts) sur la position et l'orientation du personnage principal et donc de la caméra.
- Troisième personne (*third person* ou *TPS*) : le système de gestion de la caméra suit le personnage d'intérêt depuis une certaine distance (généralement elle est située légèrement au dessus et derrière une de ses épaules). Ici le système doit réagir à la fois aux éléments du décor (afin d'éviter les occlusions et les collisions) et aux interactions de l'utilisateur (afin de maintenir à l'image les objets d'intérêt). Des problèmes apparaissent lorsque le système n'arrive pas à prendre en compte des éléments d'interactions qui font partie intégrante du jeu. Un exemple classique de mise en défaut de certains systèmes *TPS* se produit lorsque le joueur décide de faire reculer le personnage jusqu'à ce que celui-ci soit collé à un élément du décor (*e.g.* un mur). La décision prise dans ce cas est généralement de revenir à une vue frontale du personnage, provoquant ainsi une interruption dans le *gameplay* et une désorientation du joueur en ne montrant plus à l'écran la majeure partie de l'action (cf. figure 2.4). En outre, de par l'imprécision des méthodes de détection d'occlusions utilisées, il n'est pas rare que la projection du personnage principal soit partiellement voire significativement recouverte à l'écran.

- Ralentis (*Action replays*) : les ralentis sont très souvent utilisés dans les jeux de voiture et les jeux multijoueur, afin de mettre en avant des événements importants ou spectaculaires que le joueur aime revoir (cf. figure 2.5). Il est essentiel que les ralentis soient éloquents, que les éléments du jeu et leurs configurations spatiales soient immédiatement identifiable par le joueur. Il existe cependant deux types de ralentis différents : (*i*) le ralenti instantané (*in-game*) illustrant un fait marquant du jeu et (*ii*) le ralenti final permettant de revoir l'intégralité d'une course automobile par exemple.


Figure 2.5 – Capture d'écran d'un ralenti *in-game* tirée du jeu Burnout 3.

### 2.1.3 Systèmes multimodaux et de visualisation de données scientifiques

Cette section présente les exigences de contrôle de caméra dans deux catégories de système d'informatique graphique : les systèmes multimodaux et de visualisation de données scientifiques. Ces deux applications ont en commun la gestion de données hétérogènes. En effet, la multimodalité est définie comme « l'utilisation de deux ou plusieurs parmi les cinq sens en vue de l'échange d'informations » (traduction libre de Johnston [Joh03]), elle combine différents médias au sein d'un même système. Les applications de visualisation de données scientifiques quant à elles, doivent permettre l'exploration de vastes ensembles de données issues de diverses sources (systèmes d'aide à la décision, bases de données, etc.).

#### *2.1.3.1 Systèmes multimodaux*

La gestion de contenu multimodal (en particulier la mise en adéquation du langage naturel avec des images 3D) nécessite de porter une attention particulière à la coordination des modalités. Les systèmes multimodaux ont été principalement développés dans le cadre de l'éducation et de l'apprentissage. Le problème majeur relatif au contrôle de caméra dans ces systèmes, est de coordonner le choix du point de vue permettant de montrer à l'image l'objet dont il est question dans le discours.

Par exemple, une référence linguistique directe à un objet d'une scène 3D (*e.g.* « la poignée de la porte ») requiert que l'objet en question (*i.e.* la poignée) soit dans le pire des cas partiellement occulté à l'écran afin que l'utilisateur puisse le reconnaître. Pour satisfaire ce genre de contraintes de coordination, les systèmes multimodaux se sont basés sur l'utilisation intensive des points de vue par défaut (Seligmann et Feiner [SF91]) depuis lesquels des images de l'objet d'intérêt non occulté ont de très fortes chances d'être obtenues. Afin de vérifier l'éligibilité des points de vue par défaut, un simple test de lancer de rayons est mis en place (par exemple dans les travaux de Bares *et al.* [BRZL98]). Une autre classe d'approches a recours à des manipulations directes sur les polygones de la scène lors du rendu pour assurer la visibilité de l'objet d'intérêt lors du discours, *i.e.* : (*i*) soit par élimination directe de polygones des objets 3D, (*ii*) soit par manipulation du tampon de profondeur (*depth buffer*, Seligmann et Feiner [SF93]), ou (*iii*) en ayant recours à la transparence.

Au delà des simples références aux objets, la coordination entre la langue et les images pose un certain nombre d'autres problèmes intéressants en ce qui concerne le contrôle de caméra. En effet, lors du discours il ne sera pas rare de faire référence à des objets par des relations spatiales, dont la sémantique peut seulement être interprétée si l'objet est effectivement vu d'une manière appropriée. Prenons par exemple des relations spatiales entre objets telles « être devant », « être à gauche de » ou des adjectifs relatifs à la taille d'objets à l'écran comme « gros », « large ». Elles supposent toutes un placement et une orientation corrects de la caméra pour que le spectateur comprenne de quel(s) objet(s) il est question.

#### *2.1.3.2 Visualisation de données scientifiques*

Les systèmes de visualisation scientifique doivent transformer de très volumineux ensembles de données scientifiques multidimensionnels en entités 3D afin de pouvoir les afficher et ainsi découvrir les relations qui les lient. De par leur nature multidimensionnelle et à cause du volume de données qu'elles manipulent, les applications de visualisation nécessitent une aide au contrôle de caméra permettant l'exploration et la prospection des relations reliant les données affichées. En effet, les données affichées couvrent bien souvent de très vastes zones dans lesquelles une interaction directe de l'utilisateur sur la caméra est difficile. A l'heure actuelle, dans la majeure partie des applications de visualisation, l'utilisateur est très limité dans les interactions qui lui sont offertes au niveau d'un contrôle direct de la caméra pour l'exploration des données. Une automatisation du contrôle de caméra dans ces applications, ou tout au moins la possibilité d'offrir une aide à son contrôle direct est indéniablement un atout et peut grandement faciliter l'exploration de très gros volumes de données.

De ce fait, les systèmes de visualisation scientifique profiteraient énormément de techniques de contrôle de caméra capables de prendre en compte les spécificités du domaine de l'application lors de l'aide à l'accomplissement de la tâche de l'utilisateur. Toutefois, ces comportements adaptatifs requièrent l'habileté d'évaluer le potentiel perceptif d'un point de vue sur une scène, et la capacité de réagir à cette évaluation de la façon la plus bénéfique qui soit pour l'utilisateur.

### 2.1.4 Difficultés liées au contrôle de caméra

Il est possible d'identifier trois difficultés liées au contrôle de caméra, sous-jacentes aux approches présentées, à savoir :

- la gestion des sept degrés de liberté de la caméra virtuelle,
- la complexité intrinsèque des environnements 3D fortement peuplés et dynamiques,
- l'expressivité offerte à l'utilisateur pour la description d'un problème.

Tout d'abord, le contrôle direct d'une caméra virtuelle par un utilisateur demande une certaine expertise de la part de ce dernier. En effet, il est impossible de gérer à la fois les sept degrés de liberté de la caméra avec des périphériques d'entrée en possédant généralement 2 voire 3 (beaucoup plus rarement, 6 degrés de liberté sont fournis par les souris 3D). Les approches de contrôle direct des paramètres de la caméra par l'utilisateur doivent donc définir des méthodes de mappage entre les entrées fournies par l'utilisateur et les degrés de liberté de la caméra.

Une deuxième difficulté réside dans la complexité intrinsèque de la planification automatisée (partiellement ou totalement) de trajectoires de caméra. En effet, le contrôle dynamique de caméra virtuelle peut être considéré comme un cas particulier du problème de planification de trajectoire et appartient donc à la classe des problèmes PSPACE-difficiles, pour lesquels la complexité est exponentielle en nombre de degrés de liberté. Ajoutons à cette remarque que la relation entre un objet d'une scène 3D et sa projection 2D à l'écran est fortement non linéaire. Considérons un modèle de caméra basé sur les angles d'Euler (cf. section A.0.1), pour lequel les sept degrés de liberté sont : $\mathbf{q} = [x_c, y_c, z_c, \phi_c, \theta_c, \psi_c, \gamma_c]$. La projection d'un objet 3D à l'écran est définie par l'équation 2.1. On exprime cette relation comme étant un changement de base depuis le repère monde global (3D), vers le repère local de la caméra (3D) puis vers le repère image (2D). Ces changements de base sont définis par la donnée d'une matrice de rotation $R(\phi_c, \theta_c, \psi_c)$, d'une matrice de translation $T(x_c, y_c, z_c)$ et d'une matrice de projection $P(\gamma_c)$. La composition de ces trois matrices détermine les coordonnées 2D $(x', y')$ de la projection d'un point 3D $(x, y, z)$.

$$\begin{aligned} \begin{pmatrix} x' \\ y' \end{pmatrix} &= P(\gamma_c).R(\phi_c, \theta_c, \psi_c).T(x_c, y_c, z_c). \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ &= H(\mathbf{q}). \begin{pmatrix} x \\ y \\ z \end{pmatrix} \end{aligned} \tag{2.1}$$

La forte non linéarité de cette relation la rend difficilement inversible, c'est-à-dire qu'il est complexe de décider où positionner la caméra en connaissant la position d'un objet dans la scène et les coordonnées de sa projection à l'écran. De plus, cette relation n'apporte aucune information concernant l'occlusion potentielle de l'objet à l'écran.

Enfin, le but de l'automatisation du contrôle de caméra n'est pas de se cantonner à la définition de relations simples entre les objets d'une scène 3D et leurs projections et/ou dispositions à l'écran. Le contrôle de caméra doit englober la spécification de contraintes de haut niveau sur des propriétés que l'on souhaite voir à l'image. La prise en compte de propriétés visuelles significatives et de relations spatiales entre objets est essentielle dans le but d'offrir à l'utilisateur des moyens de description plus expressifs que des relations purement géométriques. Le contrôle automatique de caméra se doit d'aider les utilisateurs dans leur construction mentale des scènes 3D et leur compréhension des mondes virtuels en leur fournissant les éléments nécessaires tant au niveau spatial, temporel que causal des relations entre les objets et les événements de la scène.

## 2.2 Contrôle de caméra et cinématographie

Depuis l'apparition de la photographie, puis du cinéma, ces deux domaines artistiques ont évolué (plus ou moins) conjointement jusqu'à obtenir une convergence aussi bien au niveau des termes que des méthodes permettant de véhiculer les mêmes valeurs et significations dans l'inconscient collectif,

cf. [Aum90, Tis95]. Une fois ces règles et nomenclatures concernant les appareils photos, caméras, les notions de montage et de composition visuelle du monde réel établies, il apparaît logique de vouloir les retranscrire au contrôle et placement de caméra en environnements virtuels. La cinématographie regroupe en son sein de nombreuses notions dépassant le cadre de la gestion de caméra comme la composition visuelle, l'éclairage (photographie), la mise en scène (positionnement des acteurs et des décors), le montage, ou bien encore la gestion de la bande son ou le maquillage. Transposés aux films d'animation numériques, chacun de ces concepts devient complètement indépendant et leur gestion respective en est découplée. Toutefois, les réalisateurs de documentaires, tout comme les photographes n'ont que peu (voire pas du tout) de contrôle sur les notions de mise en scène ou d'éclairage (par exemple) qui sont complètement indépendantes de leur volonté. Il nous faut garder cette remarque à l'esprit lors de la lecture de cet état de l'art. En effet, la planification de trajectoires de caméra virtuelle en temps-réel (*e.g.* dans les jeux vidéo) correspond parfaitement au parallèle établi avec le documentaire cinématographique dans lequel le réalisateur doit présenter aux spectateurs les attitudes des acteurs sans pouvoir modifier directement leur comportement, position, orientation ou physionomie.

### 2.2.1 Le placement de caméra

Bien que la communauté cinématographique soit arrivée à un consensus concernant la meilleure façon de « faire du cinéma », il existe toujours des divergences d'opinion sur la manière d'articuler les plans et séquences entre eux. Prenons comme exemple deux points de vue différents décrits dans deux ouvrages majeurs concernant les règles cinématographiques : « La grammaire du langage filmé » [Ari76] et « The Five C's of Cinematography: Motion Picture Filming Technique » [Mas65].

Tout d'abord, présentons l'approche de Daniel Arijon [Ari76] concernant une séquence de dialogue. L'auteur catégorise les éléments de la scène en fonction du nombre de personnages principaux et énumère ensuite les positions et angles de vue adéquats permettant de respecter les contraintes fondamentales de cadrage des personnages à l'écran. Du fait de son caractère entièrement procédural, dépendant uniquement du nombre d'acteurs présents dans la scène de dialogue et donc facilement adaptable aux environnements numériques, cette approche a bien évidemment inspiré de nombreux travaux de placement automatique de caméra. En revanche, des contributions comme celle de Mascelli [Mas65] préfèrent mettre l'accent non pas sur des archétypes de scènes bien précises mais sur des principes de plus haut niveau comme la continuité temporelle, spatiale ou narrative.

Il est généralement admis que les scènes de dialogue sont filmées suivant des conventions bien définies et qu'elles peuvent donc être reproduites en environnements virtuels *via* l'application d'heuristiques retranscrivant ces règles. Arijon présente par exemple la notion de *ligne d'intérêt* qui, pour un acteur isolé est déterminée par la ligne de vue (le regard) de celui-ci, et pour un couple de personnages est symbolisée par la droite rejoignant leurs têtes. Les positions de caméra permettant alors de filmer de façon académique un dialogue sont alors standardisées par rapport à cette ligne d'intérêt.

La figure 2.6 illustre les positions de caméra aboutissant à un dialogue conventionnel. Les neuf positions identifiées par l'auteur correspondent aux plans classiques en cinématographie, à savoir les angles : opposés internes (caméras 6 et 7), opposés externes (1 et 2), perpendiculaires (3, 4 et 5) et parallèles (8 et 9). De plus, en s'assurant que les caméras ne traversent pas la ligne d'intérêt l'auteur est en mesure de garantir que le spectateur ne sera pas déconcerté par des changements de positions des acteurs à l'écran ou par des changements dans les lignes de vue/fuite. L'exemple présenté ici n'est qu'une des nombreuses configurations possibles entre deux acteurs à l'écran. Il existe en effet une multitude de cas de figure possibles dépendant de la position relatives des acteurs à l'écran (*e.g.* sont-ils proches ou éloignés l'un de l'autre), de leur orientation (*e.g.* parallèle ou perpendiculaire), de la direction de leur

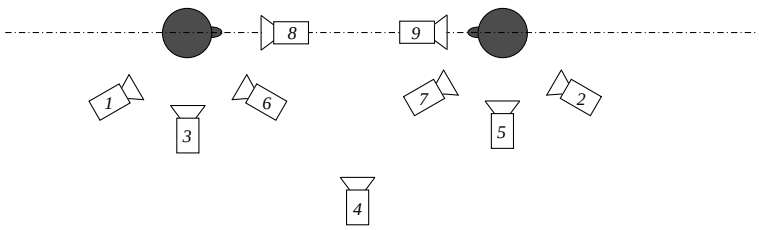



Figure 2.6 – L'idiome cinématographique pour une scène de dialogue entre deux personnages présenté par Arijon [Ari76] décrit neuf placements distincts de caméra en fonction du placement des deux acteurs et de la ligne d'intérêt.

regard (*e.g.* sont-ils face à face ou se tournent-ils le dos), ou bien encore de leur posture (*e.g.* assis, debout, allongés, etc.). En fonction de tous ces paramètres, Daniel Arijon établit une liste de points de vue conventionnels, et propose d'étendre sa méthode à trois acteurs ou plus pouvant se trouveer dans toutes sortes de configurations spatiales.

### 2.2.2 Composition visuelle

La composition visuelle permet de déterminer les arrangements spatiaux à l'écran entre les différents éléments (acteurs, décor) de la scène. La position et la distance focale de la caméra vont déterminer les types de plan de vue réalisables, lesquels peuvent être classés selon la part du personnage d'intérêt représentée à l'écran (allant de très gros plan pour une petite partie du personnage à l'écran à plein cadre lorsque celui-ci apparaît en entier, cf. section 4.2.1). Toutefois, l'orientation de la caméra est primordiale pour un placement fin des éléments de la scène à l'écran : c'est ce que l'on appelle la composition visuelle de l'image.

La composition est généralement définie en fonction des éléments à inclure dans l'image, mais aussi des lignes de fuite, des règles des tiers et des cinquièmes (lignes imaginaires découpant verticalement l'image en trois, ou cinq parties égales), des masses en présence à l'écran et, dans le cas d'un plan séquence, du mouvement des personnages dans la scène. Les images sont donc conçues par les réalisateurs dans un but précis : faire focaliser le spectateur sur un (plus rarement plusieurs) centre(s) d'attention. Celui-ci n'aura pour ce faire qu'à réaliser un balayage de l'image, dans laquelle l'arrangement des éléments aura été pensé de telle manière à la rendre inconsciemment agréable à regarder pour le spectateur en présentant clairement l'information désirée. Cette notion d'esthétique est très difficile à caractériser, et des études psychologiques comme celle de Nodinem, Locher et Krunpinski [NLK93] basées sur des expériences de suivi de regard (*eye tracking*) ont d'ailleurs montré qu'il existe des différences significatives dans la manière qu'a un individu d'appréhender (de parcourir du regard) des œuvres d'art subjectivement définies comme étant ou non visuellement équilibrées.

À un niveau plus pragmatique, Mascelli identifie un ensemble d'heuristiques aboutissant à des images visuellement bien équilibrées. Il affirme par exemple que « des lignes réelles ne doivent pas diviser l'image en parties égales », ou bien encore que « les lignes principales de l'image, qu'elles soient verticales ou horizontales, ne doivent pas être centrées » [Mas65]. L'auteur établit également une différenciation au niveau de l'équilibre d'une image, qu'il divise en équilibre formel (c'est-à-dire une composition symétrique de l'image) et informel (*i.e.* une composition asymétrique). Il est important de noter que

l'arrangement au sein d'une image ne porte pas seulement sur les objets physiques de la scène, mais prend en compte les notions abstraites comme les lignes de regard des acteurs, les lignes de tiers, les lignes de fuite ou bien encore des jeux de lumière. La composition est bien évidemment contrainte par le fait qu'elle doit amener à la création d'images cohérentes en ce qui concerne les continuités spatiales, temporelles ou narratives d'une même séquence.

Les plans séquences (unités narratives et temporelles restituées telles quelles dans un film, c'est-à-dire sans montage ou interruption du point de vue [www]) comportant d'importants mouvements des personnages ou une succession dense d'actions, posent bien évidemment des problèmes aux réalisateurs lors de la prise du vue afin de ne pas désorienter le spectateur. Toutefois, il est possible d'étendre les règles précédentes (sur la manière de filmer un dialogue par exemple) afin de prendre en compte ces spécificités. Ces difficultés vont poser de nouveaux problèmes lors de leur adaptation au contrôle automatique de caméra en environnements virtuels. Il sera alors intéressant d'étudier la manière dont les approches de contrôle de caméra virtuelle vont prendre en compte ces singularités pour tenter de les adapter aux applications cibles choisies.

Nous venons de présenter les spécificités cinématographiques du contrôle de caméra liées à la fois à la composition visuelle et au placement de caméra, ainsi que les principales applications nécessitant des méthodes automatiques de contrôle de caméra. Nous allons maintenant dresser la taxonomie des différentes approches existantes au problème de contrôle de caméra, illustrée en figure 2.1. Cette classification distingue tout d'abord les approches interactives de contrôle de caméra (*i.e.* prenant en compte les interactions utilisateur, section 2.3) , des approches réactives (*i.e.* ayant recours à une caméra « indépendante », section 2.4). Nous nous intéressons ensuite aux approches déclaratives, qui sont composées des approches algébriques (section 2.6.1), et des approches généralisées (section 2.6.2). Ces dernières seront caractérisées par les méthodes numériques utilisées dans la résolution du problème de contrôle de caméra, à savoir des méthodes d'optimisation ou d'optimisation sous contraintes . Enfin, nous concluons ce chapitre en nous intéressant aux différences d'expressivité offertes aux utilisateurs dans ces différentes classes de solutions.

## 2.3 Approches interactives

Cette section présente les approches interactives de contrôle de caméra en environnements virtuels. Nous distinguons toutefois à l'intérieur même de cette classe de solutions, deux sous-catégories, à savoir :

- les approches de « contrôle direct » de la caméra par l'utilisateur,
- les approches de « contrôle indirect » de la caméra,

### 2.3.1 Contrôle direct de la caméra

Les approches de contrôle direct établissent un mappage direct entre les paramètres de la caméra et les paramètres utilisateur provenant de périphériques d'entrée. Ware et Osborne [WO90] ont dressé un panorama des différentes métaphores existantes dans la littérature permettant cette injection, et les ont classées de la manière suivante :

- la métaphore de « l'œil dans la main » (*eyeball in hand*) : la caméra est manipulée directement tant au niveau des rotations que des translations, comme si l'utilisateur la tenait dans sa main. Les données fournies par l'utilisateur vont donc modifier directement la position et l'orientation de la caméra.

- la métaphore du « monde dans la main » (*world in hand*) : la position de la caméra est fixe, tout comme son orientation (elle « regarde » donc dans une direction donnée). Le monde sera quant à lui tourné et déplacé en fonction des actions effectuées par l'utilisateur. Les entrées vont donc modifier directement les positions et orientations du monde relativement à la caméra.
- la métaphore du « véhicule volant » (*flying vehicle*) : la caméra est alors considérée comme un avion. Les entrées utilisateur modifient directement les vitesses de translations et de rotations de la caméra.
- la métaphore du « promeneur » (*walking metaphor*) : la caméra se déplace dans l'environnement tout en restant à une distance (hauteur) constante par rapport au sol, cf. Hanson et Wernett [HW97].

Chacune de ces métaphores possède son domaine de prédilection. La métaphore du « monde dans la main » est par exemple la plus appropriée dans les cas où l'utilisateur souhaite évoluer dans un monde figé (c'est-à-dire dans lequel les objets sont immobiles). Ce type d'interaction est utilisé par exemple dans le système de modélisation de personnage humanoïde, JACK, développé par Phillips *et al.* [PBG92]. Cependant, bien que cette métaphore soit très intuitive à utiliser, elle ne permet pas de manipuler aisément des personnages situés sur des axes parallèles ou perpendiculaires au vecteur vision (fixe) de la caméra. Le système JACK permet de résoudre ce problème en repositionnant légèrement la caméra de façon automatique. Il offre en outre la possibilité à l'utilisateur de déplacer la caméra afin de faire apparaître à l'écran un objet qui ne l'était pas. Enfin, le système de contrôle de caméra incorporé à JACK est capable de fournir à l'utilisateur des positions de caméra pour lesquelles un objet d'intérêt apparaît non occulté à l'écran. Ceci est réalisé en plaçant une caméra auxiliaire (et non utilisée lors des rendus) possédant un très grand angle de vue (*fish-eye*) au centre de l'objet d'intérêt et dirigée vers la position de la caméra courante. Le tampon de profondeur est ensuite parcouru afin de déterminer des lignes de vue libres de tout obstacle.

L'approche dite d'*arcball* permettant de traiter des problèmes similaires a été proposée par Shoemake [Sho92] et consiste à définir une sphère virtuelle contenant l'objet 3D à manipuler. L'*arcball* repose sur l'utilisation des quaternions permettant la stabilisation des calculs et évite ainsi le phénomène de singularités bien connu induit par les angles d'Euler (*gimbal lock*) lorsque l'on effectue des rotations successives autour d'un objet tridimensionnel. Le lecteur intéressé par une étude complète des différentes solutions liées aux mappages d'entrées provenant de périphériques 2D pour effectuer des rotations 3D est invité à se reporter à l'étude de Chen *et al.* [CMS88].

Khan *et al.* [KKS*05] ont proposé une méthode d'interaction permettant l'étude proximale d'objets 3D tout en évitant automatiquement les collisions avec la caméra. Le système HOVERCAM tente de maintenir la caméra à la fois à une distance fixe de l'objet d'intérêt et dans une orientation normale par rapport à sa surface. Ainsi, autour des surfaces courbes, la caméra suit parfaitement la surface à une distance fixe; alors qu'autour d'angles, la caméra tourne de manière lisse et régulière avant de se déplacer vers une surface voisine. Enfin, autour des surfaces planes, la caméra effectue simplement une translation afin de garder l'objet d'intérêt en face du point de vue, comme le montre la figure 2.7. Des méthodes permettant de gérer les cavités *intra*-objet ainsi que les changements soudains de direction sont également proposées.

HOVERCAM repose sur l'application d'un algorithme de recherche de point le plus proche (*closest point*) d'un ensemble de polygones (un modèle 3D). Afin d'améliorer les performances de l'algorithme de recherche du point le plus proche d'un modèle, les auteurs proposent l'utilisation de distances focales fixes pour la caméra et d'un arbre de sphères englobantes approximant l'objet à des granularités différentes (cf. figure5.6, page 152). Plus nous descendons dans l'arbre, plus la représentation est fine, plus nous sommes proches de la racine, plus elle est grossière. L'algorithme consiste en une traversée de la racine vers les feuilles (*top-down*) de l'arbre. À chaque niveau, il faut déterminer si une sphère peut être

contenue dans le champ visuel fixe (les cônes de la figure 2.7) de la caméra. Si ce n'est pas le cas, la sphère est éliminée ainsi que tous ses descendants dans l'arbre. La descente continue jusqu'à obtention d'une liste des sphères les plus petites satisfaisant la contrainte d'appartenance au champ visuel de la caméra. Les polygones englobés dans ces sphères sont alors soumis à un algorithme classique de détection de point le plus proche.

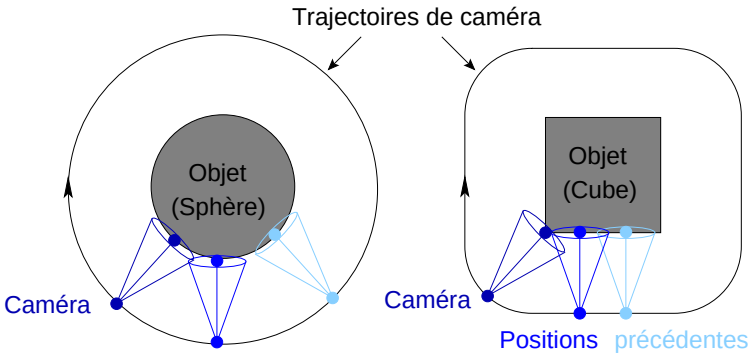



Figure 2.7 – Mouvement généré par le système HOVERCAM proposé par Khan *et al.* [KKS*05].

Deux approches analogues ont été ensuite proposées par Burtnyck *et al.*, respectivement baptisées STYLECAM [BKF*02] et SHOWMOTION [BKFK06], toutes deux également dédiées à l'inspection proximale d'objets et vouées à être intégrées dans les modeleurs 3D. Ces approches partent du constat que les applications de CAO traditionnelles ont recours à des vues prédéfinies lors de l'examen ou de l'évaluation d'objets modélisés. Cela aboutit le plus souvent à des points de vue de piètre qualité ne permettant pas la réalisation de la tâche souhaitée (l'évaluation d'un modèle). Le système STYLECAM définit donc des surfaces 3D contraignant les positions possibles de caméra. Ces surfaces sont définies de manière à assurer que les points de vues leur appartenant proposent à l'utilisateur des images satisfaisantes pour l'évaluation de l'objet d'intérêt. Des transitions automatiques sont ensuite générées pour établir les déplacements entre les différents points de vue assurant ainsi le caractère lisse des mouvements de caméra. Le système SHOWMOTION [BKFK06] reprend les idées présentées dans l'approche STYLECAM en y intégrant une interface utilisateur permettant de naviguer entre les différents points de vue proposés par le système et d'effectuer interactivement des variations dans les positions de caméras offertes par le système.

La métaphore du « véhicule volant » (*flying vehicle*) a été largement utilisée dans bon nombre d'applications d'informatique graphique. Elle est considérée comme une manière intuitive et efficace d'explorer de grands environnements 3D, comme ceux rencontrés dans les applications de visualisation de données scientifiques par exemple. Un des problèmes majeurs liés à l'implémentation de cette métaphore concerne le phénomène « perdu dans l'espace » (*lost in space*) que l'utilisateur peut rencontrer lorsqu'il doit manipuler de trop nombreux degrés de liberté dans un environnement surpeuplé, ou au contraire dans un vaste environnement n'offrant que très peu de repères visuels. Ce problème est généralement résolu en diminuant le nombre de degrés de liberté laissés à la charge de l'utilisateur et/ou en appliquant à la caméra des modèles physiques lui permettant d'éviter automatiquement des objets ou contraignant ses mouvements.

L'application d'un modèle physique à un système de contrôle de caméra a été proposé par Turner *et al.* [TBGT91]. Les entrées utilisateur sont traitées comme des forces agissant sur un système masse-ressort (la caméra). Des forces de friction et d'inertie sont ajoutées afin de gérer les degrés de liberté de la caméra qui ne concernent pas directement l'utilisateur lors de ses interactions avec le système. L'approche de Turner est facilement extensible pour la prise en compte de nouvelles forces et a inspiré de nombreuses approches basées sur l'utilisation de champs de forces permettant d'influer sur les paramètres de la caméra. Ceci est possible dans une scène 3D dont on connaît la topologie, on plonge alors la scène dans un champ de vecteurs (les forces) qui vont influencer la caméra à la fois en l'éloignant des zones saturées d'objets et en l'attirant vers des objets définis comme objets d'intérêts dans la scène [HW97, XH98], cette dernière restant toutefois contrôlée par l'utilisateur.

Les approches de contrôle interactif direct de la caméra sont problématiques car l'utilisateur doit posséder une certaine expérience pour arriver à manipuler la caméra comme il le désire. Afin de soulager les utilisateurs et de simplifier le contrôle de caméra en environnements interactifs, une autre classe d'approche a été proposée : le contrôle interactif indirect.

### 2.3.2 Contrôle indirect de la caméra

Tandis que les méthodes interactives de contrôle direct de la caméra s'intéressent à faciliter la gestion des différents paramètres de cette dernière par l'utilisateur, elles ne se préoccupent pas de contrôler précisément le mouvement des objets à l'écran. Ceci est accompli par les méthodes interactives dites de contrôle *indirect* de la caméra.

La technique de contrôle de caméra à travers la lentille (*through-the-lens camera control*) proposée par Gleicher et Witkin [GW92] permet à l'utilisateur de spécifier directement les positions souhaitées d'objets d'intérêt à l'écran. Cette approche constitue le premier effort en vue d'un contrôle indirect de caméra. À partir des positions des objets à l'écran, un calcul est effectué permettant d'inférer les paramètres de la caméra répondant au problème posé. La différence entre les positions actuelles et désirées de chaque objet à l'image est traitée comme une vitesse $\dot{\mathbf{p}}$, *i.e.* la dérivée temporelle d'une position. La position d'un point à l'écran étant complètement dépendante des paramètres de la caméra (représentés par un vecteur $\mathbf{q}$), il est possible d'exprimer la relation liant $\dot{\mathbf{p}}$ et $\dot{\mathbf{q}}$ en fonction d'une matrice Jacobienne $J$ représentant la composition des matrices de transformations, de perspectives et de projection :

$$\dot{\mathbf{p}} = J\dot{\mathbf{q}} \tag{2.2}$$

Le vecteur $\dot{\mathbf{p}}$ étant fourni par l'utilisateur (*via* une interface par exemple), Gleicher et Witkin proposent de résoudre le problème d'optimisation suivant afin de déterminer les paramètres de la caméra :

$$E = \frac{(\dot{\mathbf{q}} - \dot{\mathbf{q}}_0) \cdot (\dot{\mathbf{q}} - \dot{\mathbf{q}}_0)}{2}$$

Ce problème a pour objectif de minimiser l'énergie ($E$) représentant les modifications des paramètres de la caméra ($\dot{\mathbf{q}}_0$ étant la vitesse de la caméra à l'instant précédent). Les auteurs souhaitent donc minimiser les modifications du mouvement de la caméra, afin d'assurer une cohérence à l'écran. Ce problème de minimisation d'énergie peut être transformé en une équation Lagrangienne :

$$\frac{dE}{d\dot{\mathbf{q}}} = \dot{\mathbf{q}} - \dot{\mathbf{q}}_0 = J^T\lambda$$

L'équation précédente représente simplement le fait que le gradient de $E$ doit être une combinaison linéaire des contraintes du problème, c'est-à-dire qu'il doit minimiser les modifications de la caméra, *i.e.*

$(\dot{\mathbf{q}} - \dot{\mathbf{q}}_0)$ et assurer la position voulue à l'écran ($J^T\lambda$). La résolution de cette équation permet de calculer les valeurs du vecteur $\lambda$ (de taille 2) des multiplicateurs Lagrangiens.

À partir des équations précédentes, nous pouvons déterminer les dérivées des paramètres de la caméra, c'est-à-dire :

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}_0 + J^T\lambda$$

Enfin, une fois calculé le vecteur vitesse $\dot{\mathbf{q}}$, nous pouvons en déduire les modifications correspondantes sur le vecteur $\mathbf{q}$ des paramètres courants de la caméra. Pour ce faire, une intégration d'Euler est utilisée afin d'approximer $\mathbf{q}$ en fonction de sa dérivée :

$$\mathbf{q}(t + \Delta t) = \mathbf{q}(t) + \Delta t\, \dot{\mathbf{q}}(t)$$

L'approche de Gleicher et Witkin est extensible au contrôle de $m$ points à l'écran. Pour ce faire, il suffit de combiner les $m$ équations obtenues par la relation 2.2 en une seule en concaténant les matrices de dérivées afin de former une matrice de taille $2m * n$ ($n$ étant le nombre de degrés de liberté de la caméra) et d'assembler toutes les vitesses de chacun des points dans un vecteur de taille $m$. Une fois ces concaténations effectuées, le processus de résolution se déroule comme indiqué précédemment jusqu'à obtention des $2m$ valeurs du vecteur $\lambda$. Les auteurs proposent d'autres fonctionnalités comme le déplacement dynamique de points à l'écran par l'utilisateur, ou bien encore le suivi de point mobile.

L'approche *through-the-lens* a inspiré de nombreux travaux de contrôle indirect et son expression a été améliorée par Kung, Kim et Hong [KKH95] en ayant recours à une matrice Jacobienne simplifiée. La résolution des équations est effectuée plus efficacement par une méthode des moindres carrés liée à une méthode de décomposition en valeurs singulières (*singular value decomposition* — SVD [wtw]) afin d'éviter les problèmes de stabilité numérique induits par l'utilisation de l'intégration d'Euler. L'avantage principal lié à cette matrice pseudo-inverse issue de l'application de la méthode SVD est qu'elle fournit dans tous les cas la solution qui offre le moins de variations par rapport au vecteur $\mathbf{q}$.

## 2.4 Approches réactives

Les approches précédentes fournissent à l'utilisateur un ensemble de méthodes lui permettant de manipuler plus facilement une caméra virtuelle. Toutefois, elles ne peuvent réaliser directement de tâches de haut niveau, comme le suivi de cibles mobiles à l'écran. En effet, c'est à l'utilisateur de contrôler la caméra lui même et de faire en sorte que l'objet qui l'intéresse ne sorte pas de l'écran. Ces tâches visuelles sont pourtant souhaitables dans bon nombre d'applications graphiques. Les méthodes réactives, principalement issues de travaux en robotique, sont au contraire très adaptées à la réalisation de tâches portant sur des propriétés définies directement sur l'image (p. ex. le *target tracking*).

En robotique, les techniques dédiées au suivi de cible mobile à l'écran, Espiau *et al.* [ECR92] par exemple, sont appelées méthodes de contrôle de caméra basé image (ou bien encore asservissement visuel – *visual servoing*) et ont été largement employées pour répondre à ce problème. Le contrôle de caméra basé image nécessite la spécification d'une tâche à accomplir pour la caméra (généralement suivre un objet ou le positionner à l'écran) et permet l'inférence de caractéristiques visuelles dans la séquence d'images produite par la caméra.

Courty et Marchand [CM01] ont proposé une méthode de contrôle de caméra basé image permettant de spécifier des contraintes sur les trajectoires de caméra et de gérer ainsi une séquence d'animation complète. En plus d'une tâche principale $\mathbf{e_1}$ de positionnement d'un objet à l'écran, et si celle-ci ne contraint pas tous les degrés de liberté de la caméra, l'utilisateur peut alors définir un ensemble de contraintes secondaires.

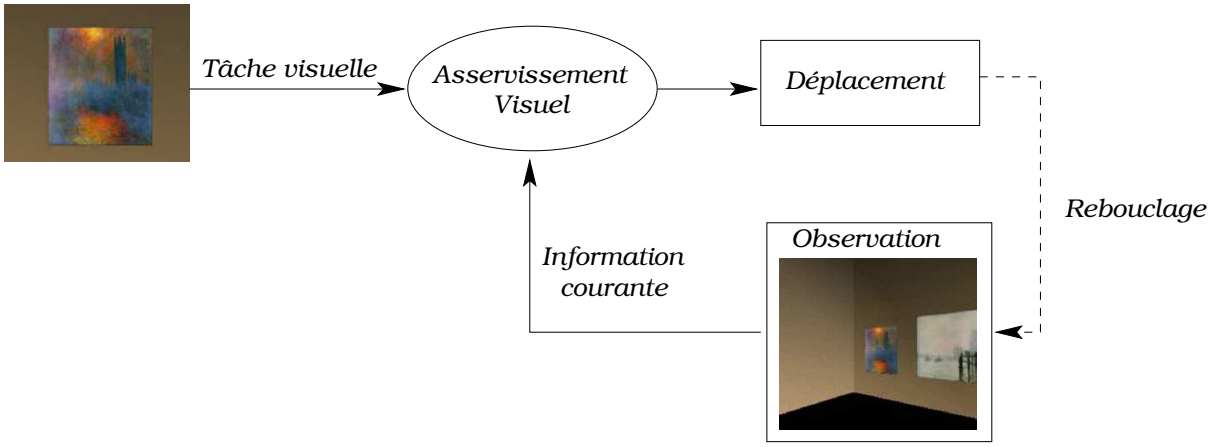



Figure 2.8 – Boucle fermée de l'asservissement visuel, d'après Courty [Cou02].

Le principe des approches d'asservissement visuel consiste en la définition d'une boucle fermée (cf. figure 2.8) ayant comme paramètres les informations perçues directement dans l'image. L'utilisateur définit ensuite dans cette boucle une configuration d'écran à atteindre, c'est la tâche visuelle à accomplir. Notons $\mathbf{P}$ l'ensemble des caractéristiques visuelles nécessaires à la réalisation de la tâche, et $\mathbf{P}_d$ la configuration désirée dans l'image. Le principe de résolution de la boucle d'asservissement visuel consiste à faire converger les valeurs de $\mathbf{P}$ vers les valeurs désirées $\mathbf{P}_d$. Dans les approches de contrôle de caméra basé image, les interactions corrélant le mouvement de l'objet d'intérêt à celui de la caméra est exprimé par le biais d'une matrice Jacobienne, la matrice d'interaction (ou Jacobien image) $\mathbf{L_P}$. Cette relation est définie par l'équation suivante, d'après Espiau *et al.* [ECR92] :

$$\dot{\mathbf{P}} = \mathbf{L_P T_c} \tag{2.3}$$

dans laquelle $\dot{\mathbf{P}}$ représente le mouvement de l'objet dans l'image attribuable à la vitesse $\mathbf{T_c}$ de la caméra (ou torseur cinématique).

La tâche visuelle principale $\mathbf{e_1}$ est définie quant à elle par :

$$\mathbf{e_1} = \mathbf{C}(\mathbf{P} - \mathbf{P_d}), \tag{2.4}$$

où $\mathbf{C}$, appelée matrice de combinaison est choisie de telle sorte que $\mathbf{CL_P}$ soit une matrice de rang plein le long de la trajectoire de l'objet suivi. Dans le cas où la tâche visuelle $\mathbf{e_1}$ contraint les six degrés de liberté de la caméra (ces approches ne considèrent pas plusieurs valeurs possibles pour la distance focale de la caméra), nous avons : $\mathbf{C} = \mathbf{L_P^+}$, avec $\mathbf{L}^+$ représentant la pseudo-inverse de la matrice $\mathbf{L}$. Le calcul de $\mathbf{L_P^+}$ est classiquement effectué par application d'une méthode de décomposition en valeurs singulières (SVD).

Afin d'assurer une décroissance exponentielle de l'erreur $(\mathbf{P} - \mathbf{P_d})$ commise dans l'image courante, la vitesse de la caméra est définie par la relation $\mathbf{T_c} = -\lambda \mathbf{e_1}$ où $\lambda$ représente un scalaire positif réglant la vitesse de convergence exponentielle de $\mathbf{P}$ vers $\mathbf{P_d}$.

Si la tâche visuelle $\mathbf{e_1}$ ne contraint pas tous les degrés de liberté de la caméra, alors il est possible d'ajouter des contraintes secondaires au problème. Dans ce cas, la matrice $\mathbf{C}$ est définie par $\mathbf{C} = \mathbf{WL_P^+}$

et la fonction de tâches est alors définie comme suit :

$$\mathbf{e} = \mathbf{W}^{+}\mathbf{e_1} + (\mathbf{I_n} - \mathbf{W}^{+}\mathbf{W})\mathbf{e_2} \tag{2.5}$$

où :

- $\mathbf{W}^{+}$ et $\mathbf{I_n} - \mathbf{W}^{+}\mathbf{W}$ sont deux opérateurs de projection garantissant la compatibilité entre le mouvement de caméra lié à la réalisation de la tâche secondaire $\mathbf{e_2}$ et la convergence de $\mathbf{P}$ vers $\mathbf{P_d}$. La matrice $\mathbf{W}$ est choisie comme étant de rang plein et telle que $\mathsf{Ker}(\mathbf{W}) = \mathsf{Ker}(\mathbf{L_P})$. Cette relation induit que $\mathbf{L_P^T}(\mathbf{I_n} - \mathbf{W}^{+}\mathbf{W})\mathbf{e_2} = 0$, et assure que la réalisation de la tâche secondaire $\mathbf{e_2}$ n'aura aucune incidence sur celle de la tâche primaire $\mathbf{e_1}$.
- $\mathbf{e_2}$ est une tâche secondaire généralement exprimée sous la forme d'une fonction à minimiser, sous la condition que $\mathbf{e_1}$ soit réalisée.

La loi de commande de la caméra sera donc donnée par la relation suivante, d'après Espiau *et al.* [ECR92] :

$$\mathbf{T_c} = -\lambda\mathbf{e} - (\mathbf{I_n} - \mathbf{W}^{+}\mathbf{W})\frac{\partial \mathbf{e_2}}{\partial t} \tag{2.6}$$

L'utilisation de tâches secondaires permet de résoudre des problèmes non-triviaux de contrôle de caméra. En effet, Marchand *et al.* [MH98, MC02] ont proposé la définition d'objectifs secondaires tels que le suivi dynamique d'un deuxième objet d'intérêt, le suivi de trajectoires, l'évitement d'obstacles ou d'occlusions. La prise en compte de tâches cinématographiques a également été implémentée, en incluant les notions de gestion de trajectoires de caméra (par *travelling* ou autres déplacements classiques en cinéma), ou bien d'optimisation d'éclairage d'un objet par exemple.

Nous détaillons ici un exemple de tâche secondaire d'évitement d'obstacles *via* la définition d'une fonction de coût à minimiser. Pour plus d'informations sur la définition de tâches secondaires et les fonctions de coût associées, le lecteur est invité à se référer à la thèse de Nicolas Courty [Cou02]. La fonction à déterminer pour réaliser l'évitement d'obstacles par la caméra consiste en une fonction de coût maximal (si possible infini) lorsque la distance entre la caméra et un obstacle est nulle. La fonction $h_s$ correspondante est alors définie comme suit :

$$h_s = \alpha\frac{1}{2\|C - O_c\|^2} \tag{2.7}$$

où $\alpha \in \mathbb{R}$, $C(0,0,0)$ est la position de la caméra et $O_c(x_c, y_c, z_c)$ est le point de l'obstacle le plus proche de la caméra, $C$ et $O_c$ étant tous deux exprimés dans le repère de la caméra.

Par extension, il est possible de définir une fonction de coût prenant en compte des obstacles multiples :

$$h_s = \sum_i \alpha\frac{1}{2\|C - O_{c_i}\|^2} \tag{2.8}$$

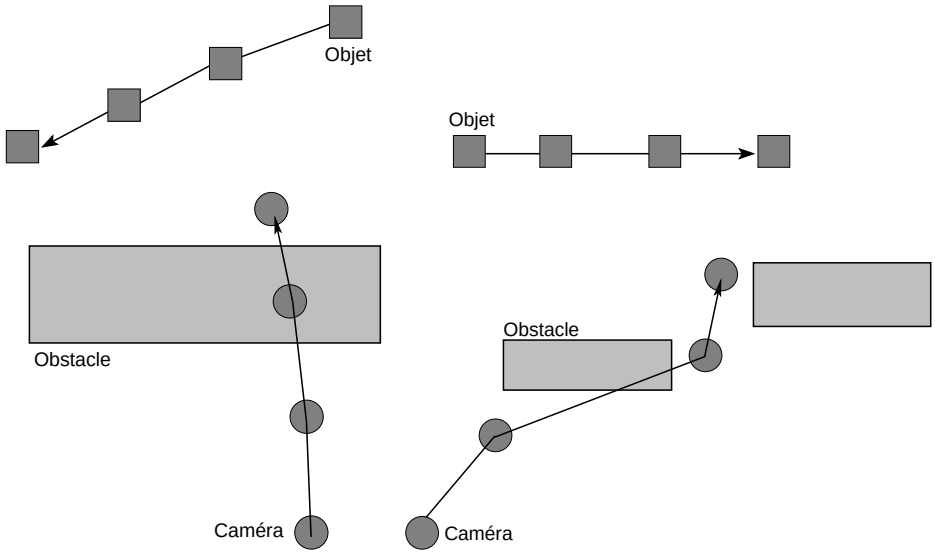
Les approches basées sur les techniques d'asservissement visuel sont algorithmiquement très efficaces et donc très adaptées à des applications fortement dynamiques comme les jeux vidéo. Cependant, il n'est pas possible de déterminer par avance le nombre de degrés de liberté qui vont être contraints par la réalisation de la tâche visuelle $\mathbf{e_1}$, et donc de savoir si la réalisation de tâches secondaires est possible. De plus, afin de maintenir des trajectoires lisses de caméra, il est nécessaire d'appliquer un processus permettant de limiter les modifications soudaines de vitesse et d'orientation de la caméra issues de la réponse au mouvement de l'objet suivi.

Une approche de contrôle dynamique de caméra spécialement étudiée pour l'application aux jeux vidéo a été proposée par Halper *et al.* [HHS01]. Cette méthode permet la prise en compte de nombreuses

propriétés comme la taille des objets à l'écran, les angles de vue, la visibilité d'objets, les hauteurs à l'écran, etc. Ces propriétés sont traitées comme des contraintes continuellement réévaluées, au contraire d'approches réactives basées sur l'application pure et dure des contraintes (citons Bares et Lester [BL97] par exemple). Ces dernières induisent des phénomènes de « sautillement » de la caméra (phénomène de *jumpiness*) dûs au fait que la caméra change brusquement de position et d'orientation afin de se diriger dans les zones solutions. Cela se produit particulièrement lors d'évitements d'obstacles ou d'occlusions.

Afin de résoudre ce problème, Halper *et al.* proposent de maintenir une cohérence visuelle (*frame coherence*) quitte à ne pas satisfaire toutes les contraintes du problème à tout instant. Pour ce faire, les solutions du problème sont calculées incrémentalement à partir des solutions courantes. Comme toutes les contraintes ne peuvent être satisfaites à tout instant, des résultats approximatifs sont obtenus en traitant les contraintes dans un certain ordre, puis en modifiant les paramètres de la caméra en fonction des contraintes restantes. Le résultat final permet d'approximer au mieux les résultats souhaités tout en offrant à l'utilisateur un ensemble de contraintes hétérogènes. Ce processus *ad-hoc* résout de façon incrémentale le problème posé, relaxant si nécessaire les propriétés ne pouvant être satisfaites afin d'assurer une cohérence à l'écran.

De plus, afin d'améliorer les mouvements de caméra (rendre les trajectoires plus lisses et ainsi augmenter encore la cohérence visuelle), les auteurs proposent d'intégrer des techniques prédictives permettant de déterminer à l'avance les mouvements de la caméra. La prise en compte de ces techniques de prédiction permet d'éviter des désagréments classiques liés aux techniques de contrôle de caméra réactif, comme ceux illustrés en figure 2.9.




(a) Artefact dû à la non-vérification d'occlusion.

(b) Artefact dû au non-maintien de la cohérence visuelle.

Figure 2.9 – Mauvais choix de trajectoires de caméra obtenus par application des techniques classiques de contrôle réactif de caméra.

Au contraire des approches réactives, dont le but n'est pas de créer un mouvement consistant de caméra, mais de réagir aux événements de l'environnment, les méthodes basées sur la planification de

trajectoires tendent à générer des mouvements cohérents en assurant un déplacement lisse et continu de la caméra, lors de son évolution dans des environnements virtuels.

## 2.5 Approches à base de planification de trajectoires

Les techniques de planification de trajectoires réalisées en robotique, tout comme les travaux basés sur une adaptation de la théorie de l'électromagnétisme, ou ceux basés sur la décomposition de l'espace de recherche en cellules ou bien encore les techniques basées sur les « feuilles de route » (*roadmaps*), ont grandement inspiré les méthodes de planification en environnements virtuels. Ces trois classes de méthodes sont détaillées dans la suite de cette section. Pour la planification de trajectoire de caméra, ces techniques peuvent être dédiées soit à la navigation 3D (rechercher dans un environnement un objet précis) ou à l'exploration 3D (parcourir un environnement afin de réunir le maximum d'informations sur celui-ci).

### 2.5.1 Planification de trajectoires basée sur les champ électromagnétiques

L'idée de ces méthodes réside dans la transposition des théories issues de l'électromagnétisme et leur application aux problèmes de planification de trajectoires. La théorie de l'électromagnétisme est issue des travaux de Maxwell [ww9w] sur l'unification de théories antérieures comme l'électrostatique, l'électrocinétique ou la magnétostatique. Elle introduit le concept de champ électromagnétique, qui permet d'expliquer le comportement dynamique de particules chargées électriquement. En physique, les forces de Lorentz et de Laplace permettent d'étudier le comportement de particules chargées. Des particules chargées électriquement de la même manière (positive ou négative) vont se repousser, alors que les particules de signe opposé vont s'attirer. L'analogie avec la planification de trajectoires de caméra s'effectue au niveau de l'interaction entre ces particules chargées et des interactions entre la caméra et l'environnement.

En effet, afin de modéliser un problème de planification de trajectoires, il suffit de représenter la caméra et les obstacles de la scène 3D comme des particules chargées identiquement afin qu'elles se repoussent. Au contraire, on modélise les objets d'intérêt comme des particules de signe opposé. Cette dernière sera donc attirée vers eux tout en restant le plus loin possible des obstacles. L'algorithme de résolution classique, proposé par Latombe [Lat91], est basé sur une série de mouvements locaux suivant le principe d'optimisation de descente de gradient. Cependant, bien qu'efficace, cette méthode s'avère handicapée par le fait qu'elle s'intègre difficilement dans des environnements dynamiques, et qu'elle fait face au problème de minima locaux, inhérent à l'utilisation de méthodes d'optimisation locale.

Cette méthode a néanmoins été améliorée dans des travaux plus récents. Beckhaus *et al.* [BRS01, Bec02] ont proposé l'utilisation de champs électromagnétiques capables de prendre en compte des environnements fortement dynamiques et interactifs. La solution consiste à discrétiser la scène en la plongeant dans une grille rectangulaire régulière et à attribuer à chacun des cubes de la scène un potentiel d'attraction ou de répulsion par rapport à la caméra. La géométrie génère donc les forces des champs électromagnétiques et la caméra est simplement vue comme une particule évoluant sous l'influence de ces forces. L'avantage de cette méthode est qu'un changement de position de la caméra peut être effectué à tout instant puisque les champs de forces sont générés par le décor, ils sont donc indépendants de la position de la caméra dans la scène et ne n'ont pas à être recalculés.

Le système CUBICALPATH [BRS01] est développé selon cette méthode dans des buts de présentation d'information ou d'exploration de scènes 3D. Une particularité de ce système de contrôle de caméra est

que les champs de force induits par les objets de la scène peuvent être modifiés à la volée. Par exemple, lors de l'exploration d'une scène, un objet d'intérêt pourra perdre de son potentiel attractif en fonction du temps qu'il passe à l'écran, jusqu'à ne plus représenter d'intérêt pour l'utilisateur. Le champ de potentiel lui étant associé sera donc modifié afin de poursuivre l'exploration en se dirigeant vers un endroit de la scène contenant un autre objet d'attraction.

### 2.5.2 Méthodes de décomposition spatiale

Ces méthodes décomposent l'espace de recherche en sous-régions (cellules) partageant des propriétés communes et créent un réseau d'interconnectivité entre ces différentes zones. En fonction de la tâche à accomplir et des propriétés de caque cellule, un ordre de visite des sous-régions est établit et permet une planification du déplacement au sein de l'environnement 3D.

Appliquées au contrôle de caméra, ces techniques permettent une navigation et une exploration dans la scène en fonction du réseau de connectivité entre les cellules et des propriétés utilisateur. Cette technique est à rapprocher de la technique dite des « portails » (*portals*) utilisée en particulier dans les jeux vidéo, lesquels doivent prendre en compte de très grands environnements 3D ne pouvant être tous chargés en mémoire au même moment. La technique des « portails » consiste à déterminer les relations spatiales de voisinage des sous-régions d'une scène. Un graphe est alors créé et permet de savoir étant donné la position d'un joueur quelles zones de la scène 3D doivent être placées en mémoire car susceptibles d'être visitées.

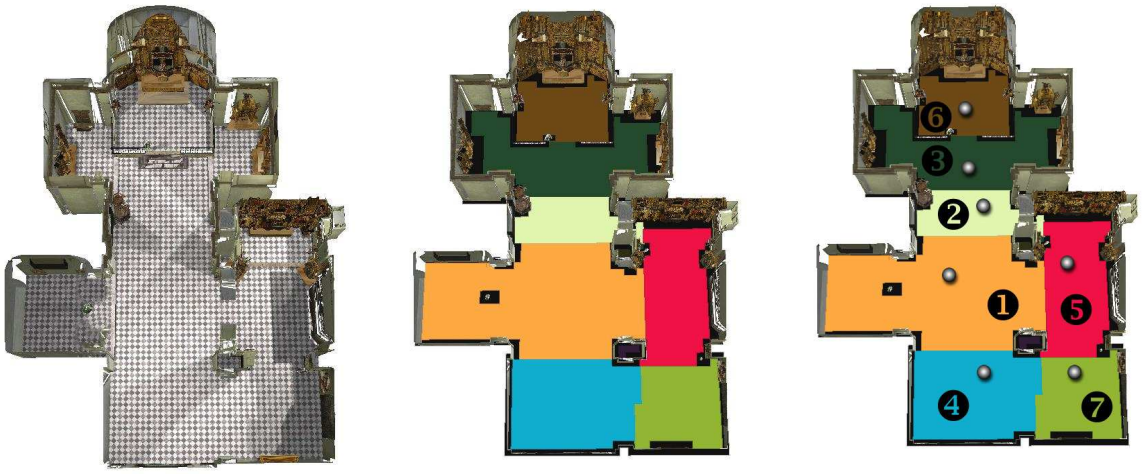
Les applications qui utilisent ces méthodes de décomposition sont par exemple les visites virtuelles. Dans ce cadre, Andùjar *et al.* ont proposé le système *Way-finder* [AVF04] afin d'améliorer le processus de navigation tout en assurant d'exposer les éléments importants de la visite virtuelle. La pertinence des régions à visiter est évaluée par une mesure d'entropie des points de vue. Pour chacune des cellules, l'algorithme évalue l'entropie des points de vue et stocke le point d'entropie maximale. Les cellules les plus pertinentes sont celles ayant l'entropie la plus élevée, et seront choisies lors de la planification de la trajectoire de la caméra.

L'entropie utilisée pour mesurer la pertinence des points de vue est celle de Shannon [Sha48]. Le score est calculé en plaçant une sphère englobante autour du point de vue d'intérêt, puis en mesurant la taille des projections des faces visibles des objets de la scène sur cette sphère. Cela permet donc d'évaluer à quel point l'information de la scène peut être capturée depuis ce point de vue. L'implémentation consiste à effectuer un rendu de la scène 3D depuis le point de vue étudié dans un tampon colorimétrique (*color buffer*, image dans laquelle chaque objet est affiché avec une couleur associée unique et sans source lumineuse). La pertinence du point de vue est alors calculée comme suit :

$$H_p = -\sum_{i=0}^{N_f} \frac{A_i}{A_t} \log \frac{A_i}{A_t},$$

où $N_f$ représente le nombre de facettes de la scène 3D, $A_i$ la surface projetée de la facette $i$, $A_t$ la surface totale des facettes projetées sur la sphère englobant le point de vue, et $A_0$ la surface projetée de l'arrière-plan (lorsqu'il existe, c'est-à-dire dans les scènes en extérieur). Dans une scène fermée (à l'intérieur d'un bâtiment par exemple), ou si le point de vue *ne voit pas* l'arrière-plan, alors $A_0 = 0$. L'entropie maximale est atteinte pour un point de vue qui *voit* toutes les facettes de la scène avec la même surface relative projetée $A_i/A_t$.

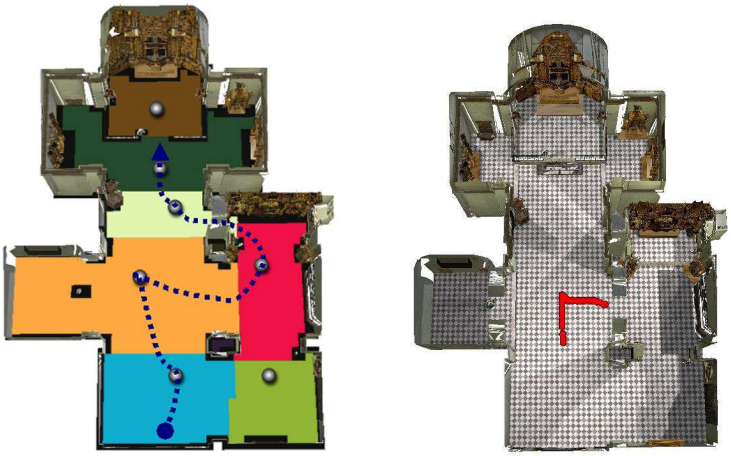
La méthode sous-jacente à *Way-finder* consiste en l'application de cinq phases successives (cf. figure 2.10) étant donné le modèle 3D d'un édifice (figure 2.10 (a)) dont nous souhaitons organiser une




(a) Vue du dessus du modèle 3D initial de l'église.

(b) Cellules détectées à l'intérieur du modèle de l'église.

(c) Ordre de visite des cellules déterminé à partir de l'entropie de chacune des cellules.



(d) Trajectoire de « haut niveau » traversant les 5 cellules les plus intéressantes.

(e) Trajectoire « bas niveau » calculée à l'intérieur de la cellule la plus intéressante.

Figure 2.10 – Illustration des différentes étapes de création de trajectoires de caméra dans une église virtuelle, par l'application de la méthode *Way-finder* proposée par Andùjar *et al.* [AVF04].

visite. La première étape consiste à générer les différentes cellules (figure 2.10 (b)) constituant le modèle grâce à un algorithme de décomposition. Il faut ensuite calculer pour chacune des cellules son entropie et établissons un ordre de visite des cellules basé sur ces mesures d'entropie (figure 2.10 (c)). Il est maintenant possible de construire un plus court chemin passant par les cellules les plus pertinentes (les visitant toutes ou seulement un sous-ensemble).

La définition de la trajectoire de caméra s'effectue ensuite en deux étapes. La première étape, dite de haut niveau, va établir dans quel ordre la caméra doit visiter chacune de cellules d'intérêt (cf. figure 2.10 (d)). Ensuite, une fois connus l'ordre et l'identité des cellules à visiter, nous pouvons définir une planification de plus bas niveau. En effet, la planification de haut niveau fournit un point d'entrée et un point de sortie pour chacune des cellules (cf. figure 2.10 (c) et (d)). Chaque cellule est ensuite décomposée spatialement (sous forme d'arbre octal, *octree*), puis un graphe d'interconnexions est crée à l'intérieur de chacune d'elle (les sommets représentent les points de vue et les arêtes les connectivités entre les points de vue). Un algorithme de type $A^*$ est ensuite appliqué une première fois afin de trouver de trouver le meilleur chemin dans une cellule, puis est réappliqué pour connecter ce chemin avec le point d'entrée et le point de sortie de la cellule (cf. figure 2.10 (e)).

### 2.5.3 Méthodes de « feuilles de route »

Les approches de planification de « feuilles de route » (*roadmaps*) sont basées sur un processus en deux phases. La première étape, hors ligne, consiste en la création d'un graphe de points de vue possibles construit par échantillonnage régulier de l'espace de recherche. Les sommets du graphe représentent des points de l'espace vérifiant une certaine propriété en fonction de l'application visée (*e.g.* absence de collision avec le décor). Ils sont reliés entre eux par des arêtes modélisant un déplacement « sûr » par rapport au problème étudié (*e.g.* une arête relie deux sommets si un déplacement de l'un vers l'autre n'aboutit pas à une collision avec le décor). Un chemin du graphe correspond donc à une solution au problème posé.

Transposés au contrôle de caméra, les sommets du graphe correspondent aux points de vue potentiels et sont reliés entre eux par des arcs modélisant une absence d'obstacles entre deux sommets successifs. Un chemin du graphe représente donc une succession de points de vue atteignables dans la scène 3D, c'est-à-dire que la caméra pourra se déplacer d'un point de vue à l'autre du chemin sans collisions avec le décor (fixe) de la scène. Les approches de « feuilles de route » probabilistes, (*probabilistic roadmaps* ou *PRM*) échantillonnent la scène non plus de manière régulière mais de manière probabiliste pour créer ce graphe. Elles ont été utilisées par exemple par Li et Ting [LT00] afin de corriger les entrées utilisateur lors d'un contrôle interactif de caméra.

Salomon *et al.* [SGLM03] décrivent une méthode permettant de créer automatiquement des trajectoires de caméra sous contraintes dédiées à la navigation d'avatars virtuels dans des environnements complexes. Leur approche construit, durant une phase préliminaire, une *roadmap* globale de la scène 3D en générant aléatoirement des positions de caméras sans collisions avec l'environnement. Les sommets de ce graphe sont connectés *via* l'utilisation d'un planificateur local qui permet d'assurer la satisfaction des contraintes de non collisions et de distance au sol (l'avatar représente un humain ou un robot se déplaçant dans la scène, le point de vue aura donc une hauteur fixe). Le graphe est ensuite traité afin d'éliminer les redondances et d'assurer que tout point de l'espace puisse être atteignable par l'avatar (sous réserve de satisfaction des contraintes).

Lors de l'exécution, l'utilisateur doit spécifier le point de départ et la position d'arrivée de l'avatar. L'algorithme parcourt alors le graphe afin de trouver un chemin consistant. L'utilisateur est alors guidé le long de ce chemin et peut décider de prendre le contrôle de l'avatar pour explorer la scène à sa guise.

Le principal point faible de cette approche est l'étape de création du graphe qui est très coûteuse à la fois en temps et en ressources. De plus les trajectoires générées sont rectilignes et peu naturelles comparées à un contrôle direct de l'avatar par un utilisateur.

Afin de résoudre ce problème, Nieuwenhuisen et Overmars [DN03] ont proposé une méthode basée sur les *PRM* permettant de générer des mouvements de caméra plus fluides. Au lieu de créer les trajectoires les plus courtes en appliquant simplement l'algorithme de Dijkstra dans le graphe de connectivité, les auteurs proposent de prendre en compte des notions de qualité lors de l'élaboration des chemins. Des virages plus larges sont préférés à ceux en épingles par exemple. Une fonction de pénalité permet donc d'évaluer la qualité des trajectoires générées, en prenant en compte les angles existants entre les arêtes du chemin.

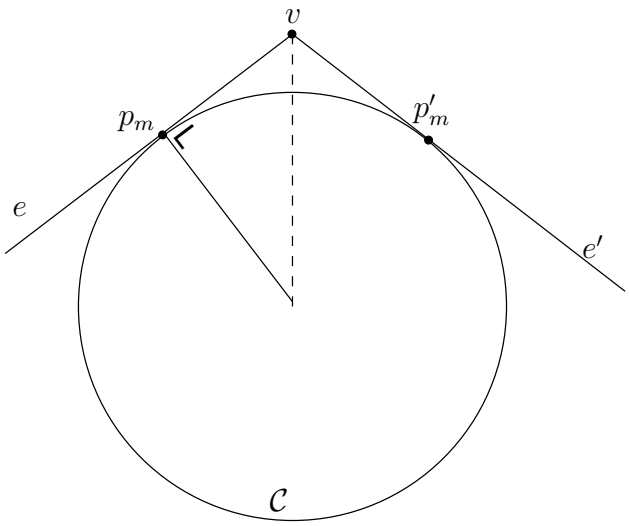



Figure 2.11 – Remplacement d'un angle saillant d'une trajectoire de caméra par un arc de cercle afin d'obtenir une courbe de continuité $C^1$.

Une fois le « meilleur plus court » chemin sélectionné, les auteurs proposent d'effacer les angles droits (arêtes de continuité $C^0$, insuffisante pour obtenir des trajectoires de caméra fluides). La méthode consiste à remplacer un morceau de la trajectoire de caméra composée initialement de deux arêtes par un arc de cercle (cf. figure 2.11). Ce dernier, tangent aux deux arêtes $e$ et $e'$, élimine la discontinuité de premier ordre située au point $v$. La répétition de cette méthode à chaque sommet du graphe permet l'obtention d'une trajectoire $C^1$. Toutefois, ceci peut introduire des collisions entre la caméra et les objets du décor, en particulier lorsqu'un obstacle se situe entre le sommet $v$ et le cercle $\mathcal{C}$. Un moyen de résoudre ce problème est de diminuer petit à petit le rayon de l'arc de cercle remplaçant le sommet $v$. En effet, il apparaît évident que si la trajectoire initiale ne contenait pas de collisions entre la caméra et le décor, alors il existe une valeur de rayon pour l'arc de cercle $\mathcal{C}$ telle que la nouvelle trajectoire soit elle aussi sans collisions avec le décor. Une heuristique est ensuite appliquée dans le but de lisser les changements de direction du vecteur vision de la caméra. Les trajectoires obtenues sont lisses (de continuité $C^1$) à la fois en ce qui concerne les mouvements et les changements d'orientation de la caméra.

Un domaine d'application des méthodes interactives de contrôle de caméra est l'endoscopie virtuelle,

qui consiste en l'exploration de certaines parties du corps humain modélisées en 3D. Des difficultés liées au contrôle interactif de la caméra apparaissent lors de l'examen de certains organes internes dont la structure est complexe. Un des prérequis de l'endoscopie virtuelle est de mettre en évidence les éléments importants des organes (tumeurs, etc.) tout en évitant les occlusions inhérentes au caractère confiné de l'environnement. Les techniques utilisées sont pour la plupart basées sur de la planification de trajectoires empruntées à la robotique [GPZT98, JLS*97], ou bien encore des adaptations de techniques de champs électromagnétiques basées sur l'utilisation de forces attractives et répulsives [HMK*97].

Ces approches se concentrent soit sur des tâches d'examens proximaux d'objets soit sur des objectifs d'exploration de vastes environnements 3D. En général ces tâches sont en exclusion mutuelle, c'est-à-dire que les applications réalisent soit l'une, soit l'autre mais rarement les deux à la fois. Pourtant, certains travaux ont été menés dans le but de permettre des transitions entre ces deux objectifs. Drucker *et al.* [DGZ92] ont proposé le système CINEMA de gestion de mouvements de caméra. Ce dernier a été conçu dans le but de combiner différentes métaphores de contrôle interactif direct de caméra (*eye in hand, world in hand, flying vehicle*, etc.). CINEMA propose un interface de définition de procédure permettant à l'utilisateur de créer ses propres métaphores de contrôle de caméra, ou bien d'étendre un jeu de métaphores prédéfinies. Zeleznik et Forsberg ont d'ailleurs démontré la pertinence de cette approche de transitions entre différents types d'interaction, en l'utilisant dans leur système UNICAM [ZF99]. Cette technique permet alors des transitions fluides de la caméra entre différents modes d'interaction proposés à l'utilisateur *via* l'esquisse de gestes réalisés à la souris.

Il existe de nombreuses manières de mapper les entrées utilisateur aux paramètres d'une caméra virtuelle. Les efforts consentis pour la définition de métaphores de toujours plus haut niveau permettant de s'abstraire du contrôle direct de ces paramètres peuvent être vus comme un premier pas vers l'automatisation complète du placement de caméra en environnements virtuels. Atteindre ce but semble toutefois difficile sans offrir aux utilisateurs la possibilité de décrire de façon non ambiguë les caractéristiques qu'il souhaite voir apparaître à l'écran. L'automatisation complète du contrôle de caméra passe donc sans aucun doute par la définition d'une représentation claire et expressive des propriétés utilisateur et par des techniques de résolution efficaces.

## 2.6 Approches déclaratives

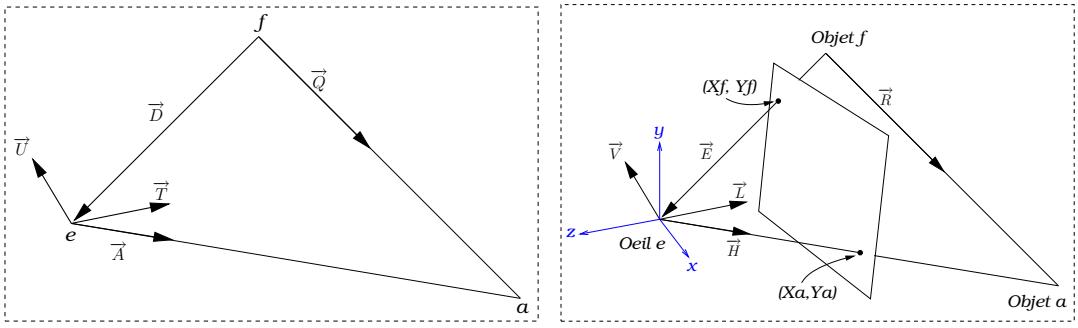
Ces approches modélisent un problème de contrôle de caméra par un ensemble de propriétés visuelles décrites par un utilisateur et que ce dernier souhaite voir vérifiées dans les images produites. Les approches déclaratives se différencient par les différentes propriétés offertes aux utilisateurs ainsi qu'aux méthodes de résolution employées. Nous distinguons dans la suite les méthodes algébriques, basées sur l'utilisation d'espaces vectoriels, des méthodes « généralisées » qui utilisent des algorithmes de résolution numériques.

### 2.6.1 Méthodes algébriques

Les systèmes algébriques sont des méthodes basées sur l'utilisation de calculs définis dans des espaces vectoriels. Appliquée aux systèmes de contrôle et de placement de caméra en informatique graphique, cette définition considére des calculs effectués sur des vecteurs définis dans l'espace image, dans l'espace monde et dans l'espace de la caméra.

Le premier exemple de système de placement de caméra basée sur une algèbre vectorielle a été développé par Jim Blinn [Bli88] lorsque ce dernier travaillait à la NASA. Son travail consistait à la

visualisation de sondes spatiales qui orbitaient autour de planètes ou de satellites. Il a donc développé un système permettant de configurer automatiquement une caméra de telle manière que les planètes et les sondes apparaissent à l'écran à des positions souhaitées. Le problème est ensuite exprimé en termes des positions des planètes, sondes et de la caméra, ainsi qu'en termes de vecteurs représentant les relations entre ces paramètres dans différents repères. La figure 2.12 illustre les représentations vectorielles du problème en fonction de la position $e$ de l'œil (la caméra), d'une planète ($a$) et d'un vaisseau spatial ($f$) dans le repère « monde », puis dans le repère de la caméra.




(a) Repère « Monde ». (b) Repère de la caméra.

Figure 2.12 – Approche algébrique développée par Blinn pour le placement de caméra.

Les paramètres nécessaires à la résolution du problème sont les coordonnées globales de la sonde spatiale $f$ et de la planète $a$ ainsi que leurs positions désirées à l'écran ($[X_f, Y_f]$ et $[X_a, Y_a]$); la donnée d'un vecteur *up* de la caméra ainsi que sa distance focale. L'avantage des approches algébriques est qu'elles peuvent être exprimées soit (*i*) en forme close (c'est-à-dire que la solution du problème est issue de la combinaison de fonctions et d'opérations mathématiques classiques) ou (*ii*) être résolues par l'application successive d'un algorithme numérique.

Une solution en forme close permet de calculer directement les valeurs des matrices de translation $T$ et de rotation $R$ nécessaires pour appliquer la transformation du repère global vers le repère de la caméra (cf. équation 2.1 de la section 2.1.4), et donc d'obtenir les paramètres de la caméra répondant au problème. Toutefois, il n'est pas rare que les problèmes amènent à l'apparition de singularités dans les équations algébriques (*e.g.* racine carrée d'un nombre négatif). Dans ce cas il est impossible de trouver une solution en forme close, et les solutions itératives numériques vont « boucler » puisque la solution sera inatteignable en pratique. Toutefois, en fixant un nombre d'itérations maximum, l'utilisation des méthodes numériques présente l'avantage de converger vers la meilleure solution disponible à l'étape $n$ et d'éviter les problèmes liés aux singularités, au prix d'une perte de précision sur la solution obtenue.

Les approches algébriques sont un moyen simple et efficace de résoudre le problème de placement d'une caméra dans une scène 3D en respectant des propriétés « simples » à l'écran. Ces propriétés doivent être définies en termes de relations vectorielles entre les objets de la scène, leurs positions à l'écran et la caméra. Cette solution algébrique pâtit de son manque d'expressivité : les objets sont abstraits en tant que points à l'écran. De plus, la solution est limitée au positionnement de deux objets à l'écran et est peu applicable en pratique.

Des approches ont néanmoins eu recours à l'algèbre vectorielle de Blinn dans des systèmes basés sur l'utilisation d'idiomes cinématographiques (configurations classiques de positionnement de sujets à

l'écran et de types de plans de vue). Christianson *et al.* [CAH*96] proposent un ensemble de plans de vue classiques définis depuis la littérature cinématographique (Arijon [Ari76]) sous forme de procédures combinées dans un arbre ET/OU décrivant la manière dont doit être filmée une scène. Les idiomes proposés référencent un ensemble de positions prédéfinies pour les acteurs à l'écran, *e.g.* lors d'un dialogue, et les positions de caméra correspondantes sont calculées *via* la méthode algébrique de Blinn. De même, Butz [But97] propose le système CATHI de génération de présentation multimédia. CATHI est basé sur une grammaire définissant les buts communicatifs à atteindre lors de la présentation et les définit en tant qu'idiomes.

Des techniques algébriques ont aussi été appliquées à la description d'œuvres 2D (peintures, tentures, etc.) et concernent les visites multimédia virtuelles, qui permettent de la mise en place d'un guide audiovisuel virtuel. Zancanaro *et al.* [ZSA03, ZRS03] ont exploré la possibilité d'avoir recours à des assistants personnels (*PDA*) pour la mise en place de systèmes de guide multimédia de musées alliant les commentaires enregistrés à la visualisation 2D des œuvres. Leur méthode calcule des déplacements de caméra sur des images immobiles (*e.g.* des fresques accrochées à un mur) afin d'attirer l'attention des visiteurs sur les éléments essentiels des œuvres, ainsi que sur des détails importants potentiellement difficiles à identifier sur la seule base du commentaire audio.

De la même manière, Palamidese [Pal96] a développé une approche algébrique de contrôle de caméra en deux étapes permettant de générer des descriptions de peintures. La première étape consiste en la définition de trajectoires de caméra qui vont mettre en lumière les détails de la peinture, puis la caméra effectue un zoom arrière afin de restituer ces détails dans le contexte global de l'œuvre ($2^e$ étape).

En conclusion, les systèmes algébriques offrent une alternative efficace au placement automatique de caméra lorsque les propriétés que l'on souhaite obtenir sur l'image sont aisément exprimables comme relations entre des points situés à l'écran. Leur application est néanmoins limitée dû à l'abstraction nécessaire des objets en tant que points. Ceci peut bien évidemment aboutir au calcul de positions de caméra fortement inadéquates aux objets impliqués dans les relations, en particulier lorsque ceux-ci ont des formes complexes ou sont particulièrement imposants. En pratique les approches algébriques sont peu utilisées à cause de leur expressivité limitée. Afin de résoudre ces problèmes, de nombreux travaux se sont penchés sur l'application de méthodes de résolution de contraintes, d'optimisation ou d'optimisation sous contraintes aux problèmes de placement et de contrôle de caméra.

### 2.6.2 Méthodes généralisées

Sous l'appellation « approches généralisées », nous regroupons les méthodes qui formalisent un problème de contrôle ou de placement de caméra soit comme un problème de satisfaction de contraintes (*CSP*), soit comme un problème d'optimisation. En pratique, bon nombre d'approches utilisent des méthodes d'optimisation sous contraintes. La figure 2.13 présente une taxonomie des différentes approches généralisées. Les propriétés spécifiées sur les images, qu'elles soient ou non cinématographiques, sont exprimées en tant que (*i*) contraintes numériques (on parlera alors de contraintes dures, c'est-à-dire que les propriétés doivent obligatoirement être vérifiées) ou (*ii*) de fonctions objectif (les propriétés doivent dans ce cas être préférentiellement vérifiées) en fonction des paramètres de la caméra et des objets composant la scène. Le processus de résolution consiste alors en l'exploration de l'espace de définition de la caméra à la recherche de configurations minimisant les fonctions objectif et/ou satisfaisant les contraintes du problème. Les approches se différencient par l'expressivité des propriétés offertes à l'utilisateur et par l'efficacité du processus de résolution numérique mis en jeu.

Il existe de nombreuses méthodes de résolution utilisables pour les approches généralisées et les solveurs sous-jacents se différencient principalement dans leur manière de gérer les problèmes sur-

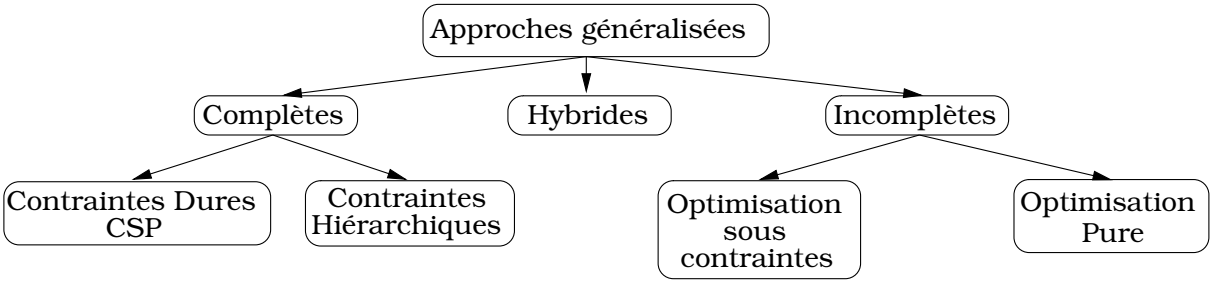



Figure 2.13 – Taxonomie des approches généralisées appliquées aux problèmes de contrôle de caméra.

contraints (l'utilisateur a spécifié un problème insoluble) et sous-contraints (la description du problème admet une infinité de solutions). Nous allons explorer ces différentes techniques en distinguant deux axes :

- la nature complète ou incomplète des solveurs,
- la nature discrète ou continue de l'espace de recherche exploré.

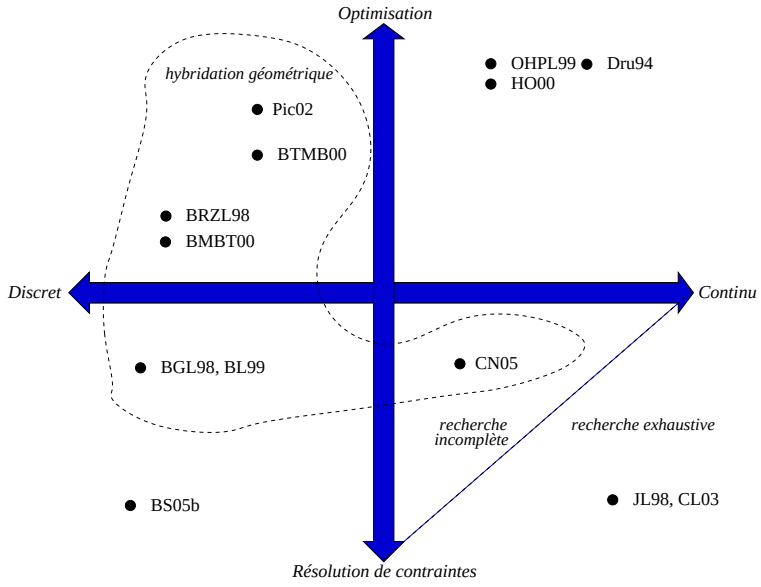



Figure 2.14 – Décomposition des approches généralisées selon deux axes : (*i*) la nature des domaines des variables et (*ii*) la nature des méthodes de résolutions employées.

La figure 2.14 présente une classification des approches généralisées présentées dans cette section. L'axe horizontal représente la nature des domaines de recherche pris en compte. Il permet la distinction des approches suivant qu'elles ont recours à un espace de recherche discrétisé, continu ou mixte. L'approche [CN05] correspond à la contribution des *volumes sémantiques*, que nous présentons dans le chapitre 4.

Les approches prenant en compte des espaces totalement discrets testent un sous-ensemble des configurations possibles de caméra établies selon un découpage régulier ou stochastique des domaines associés à chaque paramètre de la caméra. Cette discrétisation a pour but de réduire le nombre de configurations à tester, en vue d'accélérer le processus de résolution. À l'inverse, les approches continues mettent en œuvre des techniques permettant d'explorer l'ensemble des configurations possibles de caméra.

La nature du processus de résolution utilisé est illustrée par l'axe vertical de la figure 2.14. Il permet de distinguer les approches d'optimisation pure (recherche d'un minimum local ou global par descente sous gradients, etc.), de celles faisant appel à des techniques de satisfaction de contraintes. À une extrémité de cet axe nous retrouvons les approches d'optimisation pure, considérées comme des méthodes « souples », pour lesquelles la meilleure solution (éventuellement seulement localement) est calculée en minimisant une agrégation de fonctions objectif. La méthode d'agrégation la plus simple consiste à tenter de minimiser les violations de chacune des propriétés du problème. Situées à l'opposé de cet axe vertical, les méthodes de satisfaction de contraintes permettent d'assurer le respect des propriétés décrites par l'utilisateur, au contraire des méthodes d'optimisation. À ce titre les approches de satisfaction de contraintes sont considérées comme des méthodes « dures » de résolution alors que l'optimisation est une forme de résolution « souple ». À mi-chemin, nous retrouvons les approches hybrides mixant les techniques de satisfaction de contraintes avec des processus d'optimisation, plus généralement appelées techniques d'optimisation sous contraintes.

Avant de nous intéresser à la classification des différentes approches généralisées telles que différenciées dans les figures 2.13 et 2.14, nous présentons tout d'abord les différences de modélisation d'une propriété visuelle dans les approches d'optimisation et les approches de satisfaction de contraintes.

#### *2.6.2.1 Modélisation d'une propriété*

La caractérisation des propriétés offertes à l'utilisateur dans le cadre de problème de placement ou de contrôle de caméra que nous avons utilisée tout au long de ce chapitre, est motivée par des ouvrages de littérature cinématographique [Ari76, Mas65]. Ces propriétés sont formulées en tant que contraintes et/ou fonctions objectif définies sur les paramètres de la caméra et sont ensuite résolues par un processus numérique (complet ou incomplet). Nous allons illustrer la définition de contraintes et fonctions objectif pour les propriétés cinématographiques d'angle de vue et de cadrage d'un objet à l'écran (cf. section 4.2). La première consiste à filmer un objet depuis un certain angle de vue (face, profil, etc.) comme le montre la figure 4.7 située page 103. La réalisation de cette propriété peut être trivialement modélisée comme un produit scalaire entre le vecteur vision de la caméra et un vecteur représentant l'orientation de l'objet :

$$f_1(\mathbf{q}) = 1 - (\vec{\mathbf{v}} \cdot \vec{\mathbf{o}} + 1)/2$$

où $\mathbf{q} = [x_c, y_c, z_c, \phi_c, \theta_c, \psi_c, \gamma_c]$ représente le septuplet des paramètres de la caméra, $\vec{\mathbf{o}}$ un vecteur unitaire correspondant à l'orientation intrinsèque de l'objet et $\vec{\mathbf{v}}$ le vecteur vision de la caméra (obtenu en fonction de $\phi_c, \theta_c$ et $\psi_c$). Une approche purement optimisation cherche donc à maximiser $f_1$ tandis que la satisfaction de contraintes fournit un ensemble de valeurs de $\mathbf{q}$ pour lesquelles la contrainte est satisfaite.

Une version simplifiée de la propriété de cadrage d'un objet à l'écran consiste à contraindre la projection du centre $P$ de l'objet d'intérêt à une position spécifiée $(x_d, y_d)$ à l'écran. Dans le cadre d'une approche d'optimisation, la définition d'une fonction objectif correspondant à cette version simplifiée de la propriété de cadrage tend à minimiser la distance Euclidienne entre $(x_d, y_d)$ et la projection de $P(x_P, y_P, z_P)$ à l'écran. Notons toutefois qu'il faut veiller à normer la distance si nous souhaitons effectuer des comparaisons et/ou des combinaisons entre différentes fonctions objectif caractérisant les propriétés. Pour ce faire, nous pouvons introduire une fonction permettant de normaliser cet écart entre

les points, comme la fonction $\tanh(x)$ par exemple. Les approches optimisation tendent donc à maximiser la fonction objectif suivante :

$$f_2(\mathbf{q}) = 1 - \tanh\left((H(\mathbf{q}).[x_P, y_P, z_P] - [x_d, y_d])^2\right)$$

Une formulation orientée satisfaction de contraintes de cette même propriété se doit de prendre en compte une certaine flexibilité dans l'écriture de la contrainte associée, sous peine d'échecs systématiques lors du processus de recherche. Jardillier et Languénou [JL98] considèrent par exemple la propriété de cadrage comme l'appartenance de la projection de $P$ à un cadre 2D $C = [(x_1, y_1), (x_2, y_2)]$, où $x_1$ et $y_1$ (resp. $x_2$ et $y_2$) sont définis comme légèrement inférieurs (resp. supérieurs) aux valeurs de $x_d$ et $y_d$. Nous avons donc la définition suivante des contraintes associées à la propriété de cadrage simplifiée :

$$\begin{cases} (x', y') & = H(\mathbf{q}).(x_P, y_P, z_P) \\ x' & \geq x_1 \\ x' & \leq x_2 \\ y' & \geq y_1 \\ y' & \leq y_2 \end{cases}$$

#### *2.6.2.2 Méthodes d'optimisation*

Les approches de contrôle de caméra basées purement sur des méthodes d'optimisation numériques expriment les propriétés en tant que fonctions objectif à maximiser ou minimiser lors du processus de résolution. Une métrique doit être spécifiée afin de pouvoir combiner ces fonctions de façon cohérente. Les approches d'optimisation numérique peuvent être séparées en différentes catégories, à savoir :

- les approches déterministes telles que les descentes de gradient ou la méthode de Gauss-Seidel,
- les approches non déterministes, qui peuvent quant à elles être distinguées en trois sous-familles :
  - les méthodes populationnistes (*e.g.* les colonies de fourmi),
  - les algorithmes évolutifs (les algorithmes génétiques),
  - les méthodes probabilistes (*e.g.* la méthode de Monte-Carlo),
  - les méthodes de recherche locale stochastiques (comme le recuit simulé ou la recherche tabou)

La résolution d'un problème de placement de caméra par une méthode d'optimisation consiste en la recherche d'une configuration de caméra $\mathbf{q} \in \mathbf{Q}$ (où $\mathbf{Q}$ représente l'ensemble des configurations possibles de caméra) telle que $\mathbf{q}$ maximise (ou minimise suivant la définition des fonctions objectif) une fonction de coût définie comme suit :

$$F(f_1(\mathbf{q}), f_2(\mathbf{q}), \cdots, f_n(\mathbf{q})) \text{ t.q. } \mathbf{q} \in \mathbf{Q}.$$

avec $f_i : \mathbb{R}^7 \rightarrow \mathbb{R}$ fonction objectif mesurant la satisfaction de la $i^e$ propriété, et $F : \mathbb{R}^7 \rightarrow \mathbb{R}$ représente l'agrégation des fonctions $f_i$. Dans sa représentation la plus simple, $F$ est définie comme une combinaison linéaire pondérée des différentes fonctions objectif, *i.e.* :

$$F(f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_n(\mathbf{x})) = \sum_{i=1}^{n} w_i f_i(\mathbf{x}), w_i \in \mathbb{R}$$

Dans leur système CAMPLAN, Olivier *et al.* [OHPL99] ont recours à une méthode d'optimisation évolutionnaire (*i.e.* un algorithme génétique) pour résoudre un problème de placement de caméra. Un grand nombre de propriétés est mis à la disposition de l'utilisateur comme le positionnement relatif

d'objets, la gestion d'occlusion partielle ou totale ou bien encore la possibilité de définir la taille d'un objet à l'écran. La fonction de coût utilisée correspond à une combinaison linéaire pondérée des satisfactions de chacune des propriétés utilisateur. Les sept paramètres de la caméra sont codés chacun comme allèles d'un même chromosome. Une population composée de ces chromosomes est ensuite générée aléatoirement dans l'espace de recherche. Chaque individu (*i.e.* un chromosome) est évalué par rapport à la satisfaction de chacune des propriétés. À chaque itération de l'algorithme génétique, les meilleurs 90% de la population survivent et les 10% restants sont re-générés par mutation et/ou entrecroisement (*crossover*, c'est-à-dire un échange aléatoire de certaines allèles entre deux individus) des chromosomes. Cette méthode a été étendue à la génération de trajectoires de caméra par Halper et Olivier [HO00] en obtenant via l'algorithme génétique les points de contrôle d'une spline dont les points de départ et d'arrivée sont connus et définissent la trajectoire de la caméra (dont le vecteur vision est fixé). Toutefois les performances de l'algorithme génétique ne permettent pas une génération temps réel des positions et des trajectoires de caméra et ces méthodes sont donc dédiées à des applications hors-ligne.

Cette inefficacité (relative) des algorithmes génétiques, peut être réduite en modélisant et intersectant les bornes de chacune des propriétés dans l'espace de recherche initial, comme le propose Pickering [Pic02]. L'auteur propose en effet une approche déclarative permettant de réduire l'espace de recherche en éliminant les régions inintéressantes. Par exemple, pour l'objet vu de face, les régions situées derrière l'objet d'intérêt peuvent être éliminés. Lorsque plusieurs objets sont impliqués dans une propriété, ou lorsque l'on souhaite prendre en compte plusieurs propriétés, l'espace de recherche cohérent est obtenu par intersection des espaces de recherche de chacune des propriétés. Pickering, génère l'espace des régions possibles par une approche en trois étapes :

1. la construction d'un arbre *BSP* (cf. section 4.1) qui partitionne l'espace en utilisant une version modifiée de l'algorithme des volumes d'ombrage (*shadow volumes*, cf. chapitre 5) et considère les objets comme des sources lumineuses afin de gérer la notion d'occlusion,
2. la modélisation des régions possibles des propriétés utilisateur en ayant recours aux arbres octaux (*octree*) comme illustré en figure 2.15,
3. l'élagage de l'arbre octal en éliminant tous les noeuds n'appartenant pas à l'arbre *BSP* (c'est-à-dire tous ceux qui ne satisfont pas les contraintes d'occlusion).

Ces structures de données (l'arbre *BSP* et l'*octree*) sont ensuite intégrées directement dans l'encodage des allèles de l'algorithme génétique. Une référence au voxel de l'arbre octal, ainsi qu'une orientation et une distance focale sont ajoutées pour chacun des chromosomes. Chaque allèle évoluant ensuite uniquement par *crossover*, la génération de nouveaux candidats est restreinte aux régions possibles de l'espace de recherche, augmentant ainsi les performances de l'algorithme. Toutefois, comme toutes les méthodes d'optimisation, cette approche souffre de difficultés liées à une modélisation efficace des propriétés et des fonctions objectif ainsi qu'à leur agrégation au sein d'une seule fonction de coût.

#### *2.6.2.3 Méthodes de satisfaction de contraintes*

Le cadre des problèmes de satisfaction de contraintes (*CSP*) met en œuvre une approche déclarative permettant de modéliser un large éventail de contraintes et offre un panel de techniques de résolution sûres pour celles-ci. L'utilisation de solveurs basés sur l'arithmétique des intervalles permet de modéliser les solutions d'un problème de placement ou de contrôle de caméra comme un ensemble de pavés multidimensionnels (un pavé étant un produit Cartésien d'intervalles). Snyder [Sny92] dresse un tour d'horizon de l'application de l'arithmétique des intervalles en informatique graphique. Chaque inconnue du problème de contrôle de caméra correspond à un intervalle à bornes flottantes (cf. section 3.2.2)

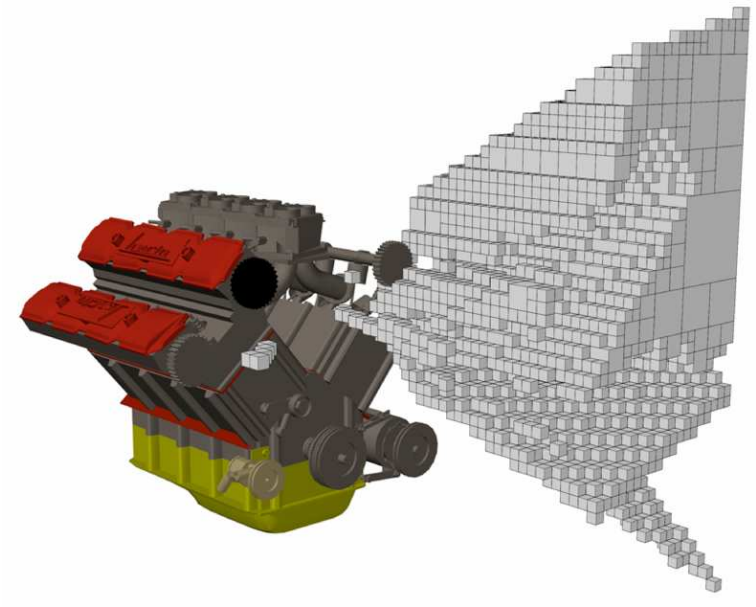


Figure 2.15 – Modélisation des régions potentiellement intéressantes de l'espace de recherche à l'aide d'arbres octaux d'après Pickering [Pic02].

qui représente l'ensemble des valeurs à laquelle cette inconnue peut prétendre. La recherche s'opère sur l'ensemble des intervalles représentant chaque variable. La résolution doit donc avoir recours à des opérateurs définis sur les intervalles plutôt qu'aux opérateurs classiques sur les nombres flottants.

Les techniques d'intervalles induisent un surcoût de par la prise en compte des intervalles par rapport aux méthodes classiques de calcul numérique. Toutefois, elles présentent un certain nombre d'avantages :

- la prise en compte native des fonctions linéaires, polynomiales, non polynomiales, non linéaires ainsi que des inégalités,
- une garantie d'approximation de la solution du problème,
- chaque région de l'espace de recherche évaluée comme solution ne contient que des solutions au problème (chaque flottant est solution),
- si aucune région n'est évaluée comme étant solution, alors le problème est garanti comme n'ayant aucune solution (il est dit inconsistant).

Jardillier et Languénou [JL98], puis Languénou *et al.* [LBGC98] proposent une méthode de satisfaction de contraintes qui construit des trajectoires de caméra en fonction de propriétés (angles de vue, cadrage, taille des objets, etc.) définies sur l'image. Le recours à des méthodes purement intervalles dans *The Virtual Cameraman* et plus particulièrement l'utilisation de contraintes universellement quantifiées permet la spécification de trajectoires de caméra satisfaisant les propriétés sur un intervalle de temps déterminé (*i.e.* $\forall t \in [t_d, t_f]$ avec $t_d$ et $t_f$ représentant les temps de début et de fin du plan séquence).

Cette approche a été améliorée par Christie *et al.* [CLG02] par l'intégration de méthodes de propagation de contraintes dans le processus de résolution, et par l'ajout de contraintes plus expressives sur les trajectoires de caméra. Celles-ci sont modélisées comme une succession de primitives cinématographiques de déplacements de caméra tels que le *travelling*, le panoramique, l'*arcing*, etc., ou de n'importe

quelle combinaison de ces mouvements. À l'inverse de la plupart des approches, qui n'assurent la satisfaction des contraintes que pour un nombre restreint de points (généralement les positions initiale et finale ainsi qu'un ensemble de points représentatifs ou *keyframes*) appartenant à la trajectoire de la caméra, les approches basées sur l'arithmétique des intervalles permettent d'assurer la satisfaction des propriétés pour tout point de la trajectoire, et donc pour la durée totale de la séquence de prise de vue. Chaque primitive de déplacement de la caméra (appelée *hypertube*) est traitée comme un problème de satisfaction de contraintes indépendant. Le problème (c'est-à-dire la suite de primitives représentée chacune par un hypertube) est résolu séquentiellement, puis le système fait coïncider les paramètres de la caméra à la fin de l'hypertube $\mathcal{H}_i$ avec ceux du début de l'hypertube $\mathcal{H}_{i+1}$.

Les auteurs ont également remplacé le processus d'évaluation (dit de bissection-évaluation) utilisé par Languénou *et al.* [JL98, LBGC98] par un algorithme en trois étapes : (*i*) résolution de contraintes, (*ii*) énumération et (*iii*) propagation. L'étape de résolution de contraintes calcule l'ensemble des boîtes intérieures au système de contraintes. Les produits Cartésiens dits intérieurs sont ceux pour lesquels tout point satisfait le système de contraintes. L'algorithme de résolution de contraintes est basé sur des méthodes de consistance locale pour des contraintes universellement quantifiées (cf. Benhamou et Goualard [BG00]). L'utilisation de contraintes universellement quantifiées permet d'assurer la fiabilité des solutions pour un intervalle de temps correspondant à la durée d'une primitive de la caméra. Une fois l'ensemble des boîtes solutions calculé, l'algorithme sélectionne la meilleure boîte évaluée par rapport à un ensemble de fonctions objectif définies pour chacune des propriétés. Enfin, la dernière phase de l'approche consiste à propager les résultats obtenus, pour un hypertube, au suivant. Ainsi la première configuration de l'hypertube $\mathcal{H}_{i+1}$ sera directement obtenue à partir des valeurs de la dernière configuration de l'hypertube $\mathcal{H}_i$. Certaines variables du problème suivant ($\mathcal{H}_{i+1}$) étant alors instanciées, l'espace de recherche est donc réduit ce qui accélère la résolution de l'hypertube $\mathcal{H}_{i+1}$.

Bien qu'assurant la *complétude*, les approches de satisfaction de contraintes ne peuvent identifier les sources d'inconsistance dans un problème *CSP*. L'utilisateur est alors obligé de retirer des propriétés de la description du problème jusqu'à obtention d'une solution. La communauté de programmation par contraintes propose un certain nombre d'approches permettant la prise en compte des problèmes surcontraints, mais au prix d'une diminution non négligeable de leur efficacité (cf. Jampel *et al.* [JFM96]).

Le cadre des contraintes hiérarchiques permet la définition de priorités entre les propriétés du problème et donc le calcul d'une solution approchée en cas d'inconsistance. Bares *et al.* ont proposé dans ce cadre le système CONSTRAINTCAM [BGL98]. Les inconsistances sont identifiées par la construction d'un graphe des paires de contraintes incompatibles (*incompatible constraint pair graph*). Si le système CONSTRAINTCAM ne parvient pas à satisfaire le problème original, il relâche les contraintes de plus faible priorité, et décompose si nécessaire un problème de placement de caméra en plusieurs sous-problèmes ayant chacun une caméra solution, puis compose le résultat en une image constituée de plusieurs vues (cf. Bares et Lester [BL99]). Le système CONSTRAINTCAM ne prend toutefois en compte qu'un sous-ensemble restreint de propriétés cinématographiques (angles et distance de vue et évitement d'occlusion) et le processus de résolution est limité à des problèmes relativement simples (*i.e.* n'impliquant que deux objets). De plus, un des inconvénients liés à l'utilisation de la relaxation de contraintes est la nécessité pour l'utilisateur d'établir une hiérarchie (c'est-à-dire une liste de préférences) des propriétés visuelles définies dans son problème. Ceci est délicat à la fois pour l'utilisateur et pour le *designer* de l'application. En effet, l'utilisateur doit prioriser les propriétés visuelles qu'il souhaite voir vérifiées dans l'image finale tout en n'ayant aucune connaissance des méthodes de résolution mises en œuvre. Pourtant, il est intuitivement plus facile de résoudre une contrainte de placement de la caméra face à un objet que de satisfaire un placement d'objet à l'intérieur d'un cadre à l'écran. Donner une plus forte priorité à une contrainte de *framing* pourra donc poser poblème lors de la résolution du problème. Au contraire,

si une préférence est attribuée de manière implicite par le designer de l'application, celui-ci doit alors classifier les contraintes afin d'éviter les inconsistances lors de la résolution. Une fois encore, la difficulté de satisfaction des contraintes est fortement dépendante des techniques de résolution employées, la hiérarchisation des contraintes est donc délicate à établir tant au niveau utilisateur qu'au niveau du programmeur.

#### *2.6.2.4 Méthodes d'optimisation sous contraintes*

Les approches d'optimisation permettent de calculer une « meilleure » solution (*i.e.* correspondant au minimum ou au maximum d'une fonction d'optimisation donnée) à un problème de placement de caméra. Toutefois, il peut arriver que la solution calculée ne satisfasse pas toutes les propriétés. Cette limitation peut aboutir à des cas pathologiques pour lesquels soit une fonction objectif domine totalement les autres dans la fonction de coût, soit chaque fonction n'est que très moyennement satisfaite. Dans les deux cas, la solution obtenue n'est que peu satisfaisante. À l'inverse, les approches basées purement sur la satisfaction de contraintes calculent l'ensemble de toutes les solutions du problème au détriment de l'efficacité du processus de résolution. De plus, elles ne permettent pas d'obtenir des solutions approchées lorsque le problème est sur-contraint. Les méthodes d'optimisation sous contraintes offrent un compromis en termes d'expressivité et d'efficacité. Le problème est modélisé comme étant à la fois un ensemble de propriétés à satisfaire (les contraintes) et un ensemble de propriétés à optimiser (les fonctions de coût à maximiser/minimiser).

Afin de pallier aux limitations de CINEMA [DGZ92], leur système procédural de contrôle de caméra, Drucker et Zeltzer ont proposé le système CAMDROID [DZ95], qui spécifie les primitives de mouvement de caméra en tant que contraintes sur les paramètres de la caméra. Les auteurs regroupent ensuite ces contraintes au sein de modules qui fournissent un moyen d'interaction de plus haut niveau avec l'utilisateur. La résolution du problème d'optimisation sous contraintes est assurée par l'utilisation de la bibliothèque *CFSQP* (*Feasible Sequential Quadratic Programming in C*) qui correspond à la combinaison d'une méthode Lagrangienne avec une recherche basée sur la méthode de Newton. Une des limitations de CAMDROID est la nécessité de définir les contraintes et les fonctions objectif comme étant continûment différentiables. Cette approche est donc limitée à l'utilisation d'opérateurs ou de fonctions différentiables, condition difficile à assurer dans bon nombre d'applications. De plus, l'algorithme est sujet aux minima locaux et est extrêmement sensible au choix du point de départ.

#### *2.6.2.5 Approches discrètes*

Les approches discrètes (par opposition aux approches continues) n'explorent pas l'espace de recherche d'un problème de contrôle de caméra en totalité, mais le discrétisent (de façon régulière ou stochastique) sur chacun des degrés de liberté de la caméra. Dans le but d'améliorer leur système CONSTRAINTCAM, Bares *et al.* [BMBT00] proposent d'avoir recours à une recherche exhaustive appliquée à une discrétisation de l'espace de recherche initial. Ils associent ensuite à chaque configuration une valeur représentant sa fidélité au respect des propriétés utilisateur. Un algorithme exhaustif d'énumération-évaluation (*generate and test*) sur cet espace de recherche est ensuite appliqué. La fonction de coût globale est exprimée comme l'agrégation des satisfactions de chaque propriété définie par l'utilisateur. L'espace de recherche initial est typiquement réduit à une grille $50 * 50 * 50$ sur chacune des dimensions $X, Y$ et $Z$ des positions de caméra, à des gradations de $15°$ pour l'orientation et à $10$ valeurs possibles pour la distance focale. L'efficacité des calculs est améliorée en limitant l'exploration à des zones potentielles de l'espace de recherche. À l'instar de Pickering [Pic02], les auteurs construisent

ces régions potentielles en intersectant des régions déterminées pour chacune des propriétés séparément. Le processus de résolution s'arrête ensuite lorsque l'évaluation d'un candidat est inférieure à un seuil de satisfaction ou lorsque la décomposition minimale de la grille est atteinte.

Comme nous le remarquons dans la section 2.7 consacrée à l'expressivité des propriétés, la traduction des propriétés en tant que relations algébriques (les seules prises en compte par les approches généralisées) peut être problématique. Nous reviendrons sur ce problème plus particulièrement dans le chapitre 5 consacré à la gestion de l'occlusion dans les approches de placement et de contrôle de caméra.

## 2.7 Expressivité

L'expressivité correspond aux différentes propriétés mises à disposition de l'utilisateur et lui permettant de placer ou de contrôler une caméra virtuelle. Elle permet donc de discriminer les approches en termes de pouvoir déclaratif. Bien qu'étant fortement corrélée à la fois à la méthode de résolution et à l'application visée, l'expressivité peut néanmoins être étudiée dans les approches de contrôle de caméra selon quatre critères principaux :

- la portée des propriétés,
- la nature des propriétés,
- le niveau d'abstraction requis sur la scène et les éléments la composant,
- l'extensibilité.

### 2.7.1 Portée des propriétés

Les propriétés sont directement liées aux entités qu'elles vont contraindre. Elles sont définies sur les paramètres de la caméra, les trajectoires de caméra ou bien encore sur les éléments à faire figurer dans l'image résultat. Les propriétés portant directement sur les paramètres de la caméra concernent les angles de vue, la distance entre la caméra et un objet ou bien encore l'évitement de collisions. La plupart des contributions [Dru94, BMBT00, HHS01, CL03] considèrent les angles de vue comme une propriété très importante, alors que la gestion des collisions n'est pas toujours prise en compte à l'exception des méthodes réactives. De même, la distance focale n'est que rarement évoquée. Elle est, en pratique, soit directement déterminée par l'application [Bli88, HCS96], soit assignable par des contraintes [Dru94, CL03], ou bien comme ce sera le cas pour la plupart des méthodes généralisées (cf. section 2.6.2), calculée directement lors du processus de résolution par satisfaction des contraintes issues des propriétés.

Les propriétés définies directement sur la trajectoire de la caméra doivent prendre en compte des contraintes de plus bas niveau comme la gestion des collisions et la cohérence de trajectoire [HO00, CL03, HHS01, DN03]. Bien que concernant principalement les approches réactives [HHS01, MC02], la cohérence de trajectoire a pour but d'éviter des changements trop brusques dans la direction ou l'orientation de la caméra. Des notions de plus haut niveau peuvent également être introduites, comme par exemple la caractérisation cinématographique de la trajectoire (en termes d'idiomes) [CL03, CLDM03] ou bien encore en termes d'objectifs narratifs [BL97, HCS96]. Citons par exemple Christie et Languénou [CL03] qui proposent la modélisation d'une trajectoire de caméra comme une suite de primitives cinématographiques (panoramiques, *travelling*, etc.).

Concernant la description des résultats souhaités en termes de propriétés visuelles sur l'image finale, la bibliographie s'étend de la définition de propriétés de bas niveau telle que « l'objet doit être à l'écran » jusqu'à la mise en place de relations plus complexes mettant en œuvre les théories de composition visuelle telle que le respect de la règle des tiers *e.g.* Gooch *et al.* [GRMS01]. À un niveau

moindre d'expressivité, nous retrouvons les propriétés purement géométriques concernant la position de points ou d'objets 3D ainsi que leur disposition à l'image. Cela regroupe la position absolue d'objets [Bli88, GW92, Dru94, HCS96, HHS01], le cadrage d'objets à l'intérieur de sous-régions de l'espace écran (*frames* appartenant ou non à l'espace image, cf. section 4.2.5) [JL98, OHPL99, BMBT00] ou bien encore le positionnement relatif d'objets à l'écran (cf. section 4.2.4) [OHPL99].

Les propriétés concernant les objets ou bien les relations entre objets de la scène 3D peuvent être définies à un niveau géométrique ou sémantique. Par exemple, la notion d'orientation d'un objet (le voir de face, de dos, etc.) nécessite la prise en compte de la nature de celui-ci (référent, fonctionnalité, etc.). De plus, la cinématographie a fourni, en plus de cent ans d'existence, un vocabulaire riche et précis qui est utilisé dans les approches de contrôle de caméra virtuelle. Il est naturel pour un utilisateur de vouloir spécifier par exemple des plans américains, ou bien encore un *over-the-shoulder*. La prise en compte de ces notions cinématographiques dans des systèmes automatiques nécessite néanmoins l'augmentation de l'information purement géométrique de la scène avec des notions sémantiques. Les approches de contrôle de caméra basées sur les idiomes issus de la cinématographie ont souvent eu recours à de tels lexiques dans leur implémentation [CAH*96, HCS96]; tout comme les approches déclaratives basées sur la mise à disposition à l'utilisateur d'une grammaire et de langages de modélisation [OHPL99, HO00, CL03].

L'expressivité semesure également par la prise en compte des notions d'esthétiques et de cognition. Des travaux comme ceux de Gooch *et al.* [GRMS01] calculent de légères déformations du point de vue d'une scène afin de respecter des règles de composition de haut niveau (règle des tiers ou des cinquièmes par exemple) en considérant seulement la géométrie des objets. Les couleurs, l'éclairage et la sémantique associés aux objets sont pourtant des notions essentielles à prendre en compte pour des critères esthétiques comme les balances de couleurs, l'équilibre ou la composition. Certaines approches prennent en compte les concepts de reconnaissance d'objets et ont développé des techniques fournissant des points de vue maximisant l'affichage d'arêtes saillantes pour maximiser l'identification d'objets 3D [HHO05, LVJ05].

Aussi surprenant que cela puisse paraître, les réactions émotionnelles inhérentes à la perception d'une image ou d'une séquence de film ont été largement délaissées dans les travaux traitant du placement de caméra. Notons toutefois les travaux de Tomlinson *et al.* [TBN00] proposant un ensemble de prises de vues de caméra définies à partir d'objectifs de communication spécifiés par l'utilisateur et basés sur les émotions devant être transmises à travers les différents plans de vue.

### 2.7.2 Nature des propriétés

Nous distinguons ici les propriétés quantitatives des propriétés qualitatives. Les premières concernent des propriétés que nous pouvons exprimer en termes de valeurs numériques exactes, *e.g.* la position d'un point à l'écran ou bien encore la distance entre la caméra et un objet de la scène. Les propriétés qualitatives, quant à elles, sont soit définies à partir d'un vocabulaire spécifique (lié à la cinématographie, lorsque l'on parle de type de plan de vue par exemple) ou à des relations sémantiques entre les objets impliqués dans la propriété (*e.g.* « l'objet $A$ doit être à gauche de l'objet $B$ à l'écran »).

Toutefois, l'expressivité des propriétés est très souvent déterminée par avance par le choix des représentations des objets et de la caméra utilisés dans chacune des approches de placement de caméra en environnement virtuel, mais également par la méthode de résolution mise en œuvre.

### 2.7.3 Niveau d'abstraction des objets de la scène

Le niveau d'abstraction des objets dépend des techniques de résolution sous-jacentes et donc des ressources et/ou de la puissance de calcul disponible pour chacune des approches. Les méthodes hors-ligne peuvent se permettre d'utiliser des représentations beaucoup plus fidèles des objets, et donc plus coûteuses à gérer lors du processus de résoltuion, que celles temps réel, qui doivent travailler en temps borné. *A contrario*, l'utilisation de simplification des modèles (en ayant recours à des englobants par exemple), permettra d'améliorer grandement la vitesse de traitement des propriétés. Le choix du niveau d'abstraction est donc principalement basé sur l'objectif que souhaitent atteindre les auteurs : simplification des modèles pour une résolution plus rapide, ou bien niveau d'abstraction plus élevé augmentant le pouvoir déclaratif et facilitant la description des propriétés par l'utilisateur. Il est donc important de garder à l'esprit ce compromis entre efficacité et expressivité lors du choix du mode de représentation des objets d'une scène.

Prenons par exemple la propriété de cadrage d'un objet à l'écran : « L'objet $A$ doit appartenir au cadre $C$ à l'écran », cette propriété étant traitée dans de très nombreux travaux nous pouvons la considérer sans perte de généralités.

Cette simple propriété a été traitée de nombreuses façons différentes suivant le niveau d'abstraction de représentation des objets 3D dans les approches. Certaines études ne peuvent prendre en compte une représentation complexe des objets de la scène et les représentent donc comme des points. Traiter notre exemple reviendra donc pour ces approches à spécifier que le centre de l'objet $A$ (le point représentant $A$ dans la scène) doit appartenir au cadre $C$, c'est le cas par exemple de [Bli88, JL98]. Plus généralement, les auteurs proposent d'utiliser un niveau d'abstraction plus élevé pour les objets de la scène. Un certain nombre d'entre eux ont recours à une représentation des objets par boîtes englobantes [BMBT00, MLB02, CL03], ou par sphères englobantes [OHPL99, HO00]. Une amélioration de ces méthodes d'approximations est possible *via* l'utilisation de hiérarchies des englobants (arbres de boîtes englobantes orientées, ou arbres de sphères englobantes). Afin d'obtenir des représentations toujours plus fines des objets, des auteurs ont recours à des rendus matériels dans des tampons basses résolutions [Pic02, HHS01] qui offrent des approximations moins grossières que les boîtes ou sphères englobantes.

### 2.7.4 Extensibilité des approches

Bien que certains auteurs aient opté pour un nombre fixe de propriétés [HCS96, HHS01], un grand nombre d'approches ont quant à elles souhaité offrir aux utilisateurs un cadre formel permettant la définition de propriétés additionnelles. La plupart des approches généralisées (cf. section 2.6.2) sont en effet basées sur des techniques de résolution extensibles. Toutefois, bien que ces approches offrent un moyen simple et naturel d'extension de propriétés, ces dernières requièrent d'être exprimées en tant que relations algébriques. Ceci peut s'avérer délicat, voire problématique dans le cas de la gestion d'objets complexes, ou lors de la spécification de relations non algébriques, comme l'occlusion entre objets. Au contraire, bien que les approches basées sur l'optimisation permettent l'intégration de propriétés non algébriques, les techniques numériques sous-jacentes vont demander un paramétrage très fin des fonctions de coût et des opérateurs d'agrégation associés à ces nouvelles propriétés. Ce paramétrage est bien souvent le résultat de nombreuses évaluations empiriques et demande un effort non négligeable en comparaison des approches de résolution de contraintes.

## 2.8 Conclusion

Nous avons extrait de la littérature scientifique les différentes méthodes dédiées à la résolution de problèmes de contrôle de caméra en environnements virtuels. Nous avons mis en évidence le fait que la communauté scientifique se dirige vers une gestion de toujours plus haut niveau des propriétés visuelles spécifiées par les utilisateurs. Toutefois, il reste un certain nombre de verrous scientifiques à lever, nous avons trouvé en particulier trois axes de recherche relativement peu couverts par la littérature, et sur lesquels il nous semble important de travailler :

- la flexibilité offerte aux utilisateurs dans la spécification et la résolution de problèmes de placement de caméra ;
- la présentation de toutes les classes de solutions équivalentes et ainsi laisser le choix de la solution satisfaisante non plus au processus de résolution, mais à l'utilisateur ;
- la prise en compte de la notion d'occlusion dans les environnements temps réel dynamiques.

Tout d'abord, dans la majorité des approches, la description utilisateur d'un problème de contrôle de caméra n'est pas évidente, et peut facilement amener à la spécification d'un problème sur-contraint. Les méthodes existantes se décomposent en deux grandes familles : les méthodes de résolution de contraintes et celles d'optimisation. Les premières ne permettent la prise en compte des problèmes sur-contraints que par la mise en place de contraintes hiérarchiques, ceci nécessite la définition de priorités entre les propriétés utilisateur. Les méthodes d'optimisation (cf. section 2.6.2.2) fournissent à l'utilisateur des solutions sans garanties aucune concernant la résolution de l'ensemble des contraintes et nécessite une agrégation précise et délicate des fonctions de coût mises en jeu. Entre ces deux extrémités, les méthodes d'optimisation sous contraintes permettent d'ajouter une certaine flexibilité au cadre des CSP. Afin d'aider l'utilisateur dans la phase de description d'un problème de placement de caméra, nous proposons dans le chapitre 3 une solution basée sur la résolution de MAX-NCSP.

Ensuite, les méthodes consacrées à la gestion cinématographique, ou artistique de contrôle de caméra, *i.e.* celles permettant à l'utilisateur de définir des propriétés de haut niveau, ayant attrait à la composition visuelle, ne calculent qu'une solution unique au problème posé. Toutefois, nous avons mis en évidence la possibilité d'existence de plusieurs classes de solutions équivalentes en termes de rendu à l'image. Afin de proposer une aide à la création cinématographique, nous introduisons dans le chapitre 4 la notion de *volume sémantique*.

Enfin, nous avons insisté sur l'aspect essentiel de la prise en compte de la notion d'occlusion dans le contrôle de caméra. Peu d'approches s'y intéressent de façon approfondie. Dans le chapitre 5, nous proposons une nouvelle méthode de gestion temps-réel d'occlusion.

# CHAPITRE 3

# Une approche numérique pour le placement de caméra

« Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it. »
Eugene C. FREUDER – Journal CONSTRAINTS – Avril 1997

Ce chapitre présente une première approche à la problématique de placement de caméra en environnements virtuels. Comme décrit dans le chapitre précédent, il existe différentes familles de méthodes de résolution d'un problème de contrôle de caméra. Ces dernières offrent un éventail de techniques de résolution allant du contrôle direct des paramètres de la caméra à l'application d'approches généralisées permettant la prise en compte de propriétés de haut niveau. C'est à cette dernière famille que nous allons nous intéresser maintenant. En effet, bien qu'offrant le plus haut degré d'expressivité, les méthodes généralisées souffrent d'un certain nombre de limitations; nous avons remarqué en particulier qu'un problème de contrôle de caméra est soit résolu par (*i*) des méthodes de résolution de contraintes, (*ii*) des méthodes d'optimisation ou (*iii*) des méthodes d'optimisation sous contraintes.

L'inconvénient principal de la première catégorie (*i.e.* les méthodes de résolution de contraintes) est de fournir un cadre de résolution « dur », c'est-à-dire qu'en cas d'inconsistance du problème l'utilisateur doit retirer des contraintes du problème avant de relancer tout le processus de résolution. Le cadre des contraintes hiérarchiques permet quelque peu de remédier à ce problème, mais l'utilisateur doit alors définir des priorités sur les contraintes, ce qui, dépendant des applications visées, n'est soit (*i*) pas envisageable soit (*ii*) un travail délicat.

Les méthodes d'optimisation souffrent quant à elles de la nécessité d'agrégation des fonctions objectif qu'elles emploient. Une solution d'un problème de contrôle de caméra peut donc être de piètre qualité à cause d'une mauvaise agrégation des fonctions objectif au sein d'une fonction de coût globale permettant l'évaluation des candidats. De même l'extensibilité de ces approches est assez limitée du fait de la difficulté de définition d'opérateurs d'agrégation entre les fonctions objectif. Enfin, les méthodes d'optimisation induisent, par le biais des fonctions objectif, une notion de progressivité dans la satisfaction des propriétés. Ceci n'est pourtant pas généralisable à toutes les propriétés de contrôle de caméra. Prenons par exemple l'appartenance d'un objet à un cadre dans une image. Une solution proposant $50\%$ de la projection de cet objet dans le cadre ne peut pourtant être considérée comme satisfaisant à moitié la contrainte de *framing*. Cette dernière est une propriété binaire qui est soit satisfaite, soit ne l'est pas mais ne peut être partiellement satisfaite.

Les méthodes d'optimisation sous contraintes, bien qu'offrant plus de souplesse dans la prise en compte de la satisfaction et de la modélisation des contraintes, ne parviennent pas cependant à s'abstraire

des problèmes d'agrégation de fonctions de coût ou d'inconsistance des contraintes des méthodes citées précédemment.

Cependant, s'intéresser seulement aux méthodes de résolution d'un problème de placement de caméra n'est pas suffisant, la modélisation joue elle aussi un rôle prépondérant. Comme nous l'avons évoqué dans le chapitre 2, il est très facile pour un utilisateur de fournir aux systèmes de contrôle de caméra, sans même le vouloir, une description trop contraignante. Un sous-ensemble des méthodes présentées dans l'état de l'art permettent toutefois de retourner un résultat malgré une description inconsistante. Certaines approches, basées sur des méthodes d'optimisation, vont retourner la meilleure solution en termes d'évaluation par rapport à des fonctions objectif, d'autres, utilisant le cadre des contraintes hiérarchiques, vont simplifier le problème en retirant progressivement des contraintes jusqu'à obtenir une affectation consistante.

Toutefois, ces solutions ont leur limites. Les méthodes basées sur l'optimisation peuvent donner lieu à des solutions ne satisfaisant pas toutes les contraintes, sans pour autant que l'utilisateur en soit informé. De même, l'utilisation de contraintes hiérarchiques ou de relaxation de contraintes, demande d'établir une liste de priorités sur les contraintes ou bien fournit une solution à un problème simplifié par rapport à la description de l'utilisateur (en retirant arbitrairement des contraintes). Ces limitations persistent même dans les cas où la description est correcte et amène à un problème ayant des solutions bien définies. Les méthodes d'optimisation fournissent une seule solution au problème, même si il en existe plusieurs ayant des caractéristiques différentes. Au contraire, les méthodes de résolution de CSP, calculent toutes les solutions du problème, sans pour autant les caractériser.

Le problème de placement de caméra en environnement virtuel présente donc les exigences suivantes :

- une méthode de description du problème en termes de propriétés de bas niveau sur les paramètres de la caméra ou en tant que propriétés cinématographiques de haut niveau,
- une méthode de résolution permettant la prise en compte de problèmes pouvant avoir un nombre fini, infini ou ne pas avoir de solutions, sans avoir d'informations permettant de statuer sur la nature du problème avant la résolution,
- une méthode permettant de discriminer les solution obtenues en termes de propriétés satisfaites, afin de fournir cette information à l'utilisateur.

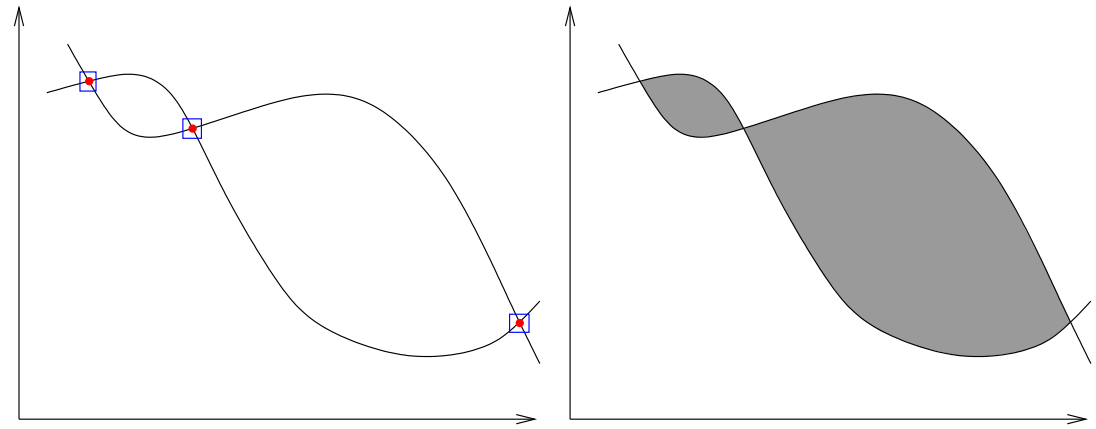
Les méthodes présentées précédemment ne permettent pas de prendre en compte toutes ces nécessités à la fois. En effet, si la plupart prennent en compte l'aspect description du problème de manière satisfaisante, aucune n'est capable de fournir à l'utilisateur une caractérisation des solutions obtenues en fonction des propriétés énoncées ou bien de prendre en compte au sein d'une même méthode de résolution à la fois les descriptions aboutissant à une infinité ou à une absence de solutions du problème.

Afin de permettre aux utilisateurs de spécifier un problème de placement de caméra comme une liste de propriétés cinématographiques de haut niveau et de prendre en compte les possibilités d'inconsistance des descriptions, nous proposons une méthode de résolution numérique continue originale. Notre solution est basée sur une approche de maximisation de contraintes numériques, *i.e.* un algorithme MAX-NCSP qui possède les caractéristiques suivantes :

- une modélisation déclarative des propriétés utilisateur permettant une extensibilité aisée,
- un processus de résolution capable de fournir à l'utilisateur toutes les solutions d'un problème ayant une description consistante et les meilleures solutions en termes de satisfaction de propriétés à un problème inconsistant,
- une caractérisation des solutions partielles en termes des propriétés satisfaites et insatisfaites de la description utilisateur.

Le cadre des MAX-NCSP permet de calculer des solutions exactes à un problème si celui-ci est *bien-contraint* (ou *sous-contraint*) et de fournir des solutions approchées en cas d'inconsistance. Notre méthode prend en compte la sémantique lors du processus de résolution afin de caractériser les solutions en termes de satisfaction (resp. insatisfaction) des propriétés utilisateur. Lors d'une inconsistance, les meilleures solutions sont fournies à l'utilisateur conjointement à la liste des propriétés satisfaites (resp. non satisfaites) du problème. De cette façon, la sémantique du problème est conservée tout au long du processus de résolution en regroupant les solutions en classes partageant les mêmes propriétés de satisfaction du problème. Cette classification des solutions du problème permet à l'utilisateur de choisir entre différentes classes de solutions équivalentes (en termes de propriétés satisfaites) lorsque le CSP est sur-contraint. Notons qu'une propriété cinématographique peut être modélisée comme une conjonction de contraintes numériques (cf. section 3.3.2) et que lors du processus de résolution nous nous intéressons à la satisfaction des propriétés et non des contraintes.

Il apparaît trivial que notre approche ne peut s'appliquer qu'à des problèmes ayant un (plusieurs) continuum(s) de solutions, comme illustré en figure 3.1. En effet, l'approche MAX-NCSP requiert le calcul d'une approximation intérieure (cf. section 3.2.6) de l'ensemble des solutions du problème, les contraintes de celui-ci doivent donc s'exprimer en tant que relations présentant un continuum de solutions (*e.g.* les inégalités).



(a) Relation ayant des solutions ponctuelles (les points entourés par des carrés).

(b) Relation présentant deux continuums de solutions (régions grisées).

Figure 3.1 – Illustration de relations permettant (b) et ne permettant pas (a) un calcul d'approximation intérieure, d'après Vu *et al.* [VSHS02].

Afin d'illustrer l'approche MAX-NCSP, nous prenons un exemple de trois relations représentées dans la figure 3.2. L'algorithme appliqué aux contraintes $a, b$ et $c$ calcule trois classes de solutions (les zones grises de la figure) car équivalentes en termes de satisfaction de contraintes (elles en satisfont toutes 2). Les approches présentées dans le chapitre 2 ne permettent pas de répondre de la même manière à ce problème :

- les méthodes de résolution de contraintes dures : ne peuvent résoudre un tel problème puisqu'il est sur-contraint, il n'existe effectivement aucun point de l'espace appartenant aux trois relations à la fois,

- les méthodes de résolution de contraintes souples : permettent de résoudre un problème sur-contraint en supprimant un certain nombre de contraintes lors de la résolution. Cette suppression s'effectue soit par rapport à une liste de priorités sur les contraintes fournie par l'utilisateur, soit par un choix arbitraire de l'algorithme.
- les méthodes d'optimisation : fournissent une solution dépendante des opérateurs d'agrégation définis pour les fonctions de coût associées aux relations. Cette solution peut appartenir à deux, à une seule ou même à aucune relation suivant l'agrégation des fonctions de coût. Nous pouvons imaginer une combinaison des fonctions de coût amenant à l'obtention d'une solution appartenant à la zone située entre les trois relations sans pour autant appartenir à aucune d'elles.
- les méthodes d'optimisation sous contraintes : suivant la caractérisation des relations en tant que contrainte ou fonction objectif, peuvent ne renvoyer aucune solution ou, à l'instar des méthodes d'optimisation, un point satisfaisant au mieux deux des trois relations sans préciser lesquelles.

La méthode MAX-NCSP est la seule à notre connaissance permettant de fournir à l'utilisateur les trois classes de solutions tout en les caractérisant par rapport aux propriétés satisfaites dans chacune des classes. Les solutions calculées pour l'exemple illustré en figure 3.2 correspondent aux trois régions grisées, chacune étant caractérisée par rapport aux contraintes qu'elle satisfait. La région grisée située en haut de l'image satisfait les contraintes $a$ et $b$, celle de gauche est consistante par rapport à $a$ et $c$, alors que celle de droite l'est par rapport à $b$ et $c$.

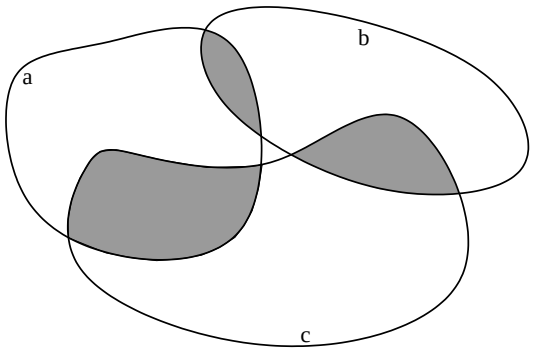



Figure 3.2 – Illustration de l'approche MAX-NCSP appliquée à trois relations $a, b$ et $c$. Les zones maximisant le nombre de contraintes satisfaites apparaissent en grisé.

Dans la suite de ce chapitre, nous spécifions tout d'abord le cadre numérique dans lequel se situe l'approche MAX-NCSP (*i.e.* les problèmes de satisfaction de contraintes et l'arithmétique des intervalles), puis nous présentons notre approche originale basée sur le cadre des MAX-NCSP pour le placement de caméra en environnements virtuels. Enfin, avant de conclure, nous proposons un ensemble de jeux d'essais permettant d'évaluer notre approche et présentons les résultats associés.

## 3.1 L'approche des problèmes de satisfaction de contraintes — CSP

La programmation par contraintes est un paradigme permettant à l'utilisateur de poser un problème sous une forme déclarative. Il ne se soucie plus de programmer la manière d'arriver au résultat, mais décrit principalement les caractéristiques du système en termes de *contraintes* devant être obligatoirement respectées (*e.g.* contraintes physiques) et éventuellement de contraintes supplémentaires qui peuvent ne pas l'être (*i.e.* des contraintes souples). Une contrainte représente une relation algébrique portant sur les variables du problème.

**Exemple 3.1 (Exemple de contrainte).** *Soit le problème géométrique suivant : « Déterminer l'ensemble des points 2D appartenant à un cercle de centre $O(x_O, y_O)$ et de rayon $r = 2$. »*
*On parlera alors de contrainte pour la relation suivante, permettant de trouver tous les points $[x, y]$ solutions de ce problème dans un espace de recherche donné :*

$$(x - x_O)^2 + (y - y_O)^2 - r^2 = 0$$

L'idée de la programmation par contraintes est d'intégrer les notions de spécification et de combinaison de contraintes au sein de relations algébriques ou logiques dans un langage de programmation de haut niveau. C'est là l'avantage de la nature déclarative de la programmation par contraintes. L'utilisateur spécifie son problème en termes de variables, de domaines de définition et de relations entre ces variables, et laisse le soin au solveur de contraintes de trouver les solutions du problème : on parle alors de *configurations consistantes*. Les algorithmes mis en œuvre implémentent les concepts de consistance entre les variables, qui permettent d'éliminer de l'espace de définition des variables toutes les valeurs inconsistantes, c'est-à-dire celles qui ne satisfont pas les contraintes. Le lecteur intéressé par un description détaillée de la programmation par contraintes est invité à se référer au livre de E. P. Tsang [Tsa93]. Par ailleurs, un excellent tutoriel consacré à la programmation par contraintes est disponible est ligne sur les pages personnelles de Roman Barták [www]. Il constitue un bon point de départ pour un novice souhaitant avoir un panorama des concepts et techniques utilisés en programmation par contraintes.

### 3.1.1 Terminologie des CSP

Le format de représentation classique de la programmation par contraintes est le format CSP pour problème de satisfaction de contraintes (*constraint satisfaction problem*). La définition d'un CSP est la suivante :

**Définition 3.1** (CSP)**.** Un problème de satisfaction de contraintes (CSP) $\mathcal{S}$ est représenté par le triplet :

$$\mathcal{S} = \langle \mathcal{C}, \mathcal{V}, \boldsymbol{D} \rangle$$

Les symboles $\mathcal{C}, \mathcal{V}$ et $\boldsymbol{D}$ ayant les significations suivantes :

- $\mathcal{V}$ représente l'ensemble des variables $\{V_1, \ldots, V_n\}$ du problème $\mathcal{S}$, elles représentent les inconnues des relations mathématiques.
- $\boldsymbol{D}$ contient l'ensemble des domaines $\{D_1, \ldots, D_n\}$ associés aux variables $\{V_1, \ldots, V_n\}$. Chacun des domaines $D_i \in \boldsymbol{D}$ contient l'ensemble des valeurs possibles pour la variable $V_i$. Ces ensembles peuvent être soit discrets (*i.e.* contenir un nombre fini de valeurs), soit continus (ils contiennent un nombre infini de valeurs, cf. section 3.2.2).

- $\mathcal{C}$ représente l'ensemble des contraintes $\{C_1, \ldots, C_m\}$ du CSP. La notion de contrainte est très large et formalise simplement une relation (numérique, logique ou symbolique) qui va limiter les valeurs possibles de certaines variables du problème (éventuellement toutes).

Nous disons qu'une variable d'un CSP est *instanciée* lorsque le système de résolution lui a attribué une valeur. Une affectation correspond à l'instanciation de certaines des variables du problème par des valeurs (de leur domaines de définition respectifs). Nous différencions les affections partielles (certaines variables du problème sont instanciées) des affectations totales ou complètes (toutes les variables sont instanciées).

Une affectation (partielle ou totale) est dite *consistante* si elle ne viole pas les contraintes du problème (*i.e.* $\nexists i$ t.q.$\neg C_i$), elle est au contraire *inconsistante* si au moins une contrainte est violée (*i.e.* $\exists i$ t.q.$\neg C_i$).

Résoudre un CSP revient donc à trouver une *affectation totale consistante*, c'est-à-dire trouver des valeurs pour chacune des variables du problème telles qu'aucune contrainte ne soit violée. Déterminer si un CSP admet une solution est dans le cas général un problème appartenant à la classe de complexité NP (Non-Déterministe Polynomial [WIKI]) pour lequel il est possible de statuer en temps polynomial sur la consistance d'une affectation totale donnée.

Le principe de résolution d'un CSP est de diminuer les tailles des domaines $D_i$ des variables $V_i$ du problème $\mathcal{S}$ par raisonnement itératif sur les contraintes $C_i$, et ce jusqu'à détection d'une inconsistance (le domaine d'une variable devient vide) ou bien jusqu'à obtention d'une affectation totale consistante.

Il existe différentes catégories de CSP, déterminées suivant le nombre de solutions admises par ce dernier, on parlera de CSP sur-contraint, sous-contraint ou bien-contraint.

**Définition 3.2** (CSP sur-contraint)**.** Un CSP $\mathcal{S} = \langle \mathcal{C}, \mathcal{V}, \boldsymbol{D} \rangle$ est dit *sur-contraint* lorsque celui n'admet aucune solution.

Dans le cadre des CSP sur-contraints, on peut s'intéresser à vouloir trouver l'affectation qui maximise le nombre de contraintes satisfaites du CSP. On se trouve alors dans le cadre des problèmes de maximisation de satisfaction de contraintes, ou encore problèmes *max-CSP*, auxquels ce chapitre s'intéressera plus particulièrement.

**Définition 3.3** (Problème max-CSP)**.** Un problème max-CSP est un problème $\mathcal{S} = \langle \mathcal{C}, \mathcal{V}, \boldsymbol{D} \rangle$ sur-contraint pour lequel on souhaite trouver une affectation maximisant le nombre de contraintes satisfaites.

À l'inverse, un problème admettant une infinité de solutions est appelé CSP sous-contraint.

**Définition 3.4** (Problème sous-contraint)**.** Un CSP $\mathcal{S} = \langle \mathcal{C}, \mathcal{V}, \boldsymbol{D} \rangle$ est dit sous-contraint s'il admet une infinité de solutions.

Enfin, un CSP n'étant ni sur-contraint, ni sous-contraint est appelé CSP bien-contraint ou CSP.

### 3.1.2 Un exemple de CSP : le sudoku

Un exemple de problème de satisfaction de contraintes illustrant clairement les mécanismes de modélisation et de résolution d'un CSP sont les grilles de sudoku. Cette section est consacrée à une présentation de l'approche CSP en nous appuyant sur l'exemple de résolution d'une grille de sudoku (celui-ci nous permettra d'instancier les concepts présentés ci-avant et d'en introduire d'autres de façon naturelle).

**Le sudoku** Une grille de sudoku est un carré composé de 81 cases, divisé en 9 sous-blocs (ou sous-carrés) de 9 cases chacun, comme illustré en figure 3.3. Certaines des cases de cette grille contiennent un nombre, d'autres sont vides. Le principe du jeu est de remplir les cases vides en utilisant des nombres de 1 à 9 en respectant les contraintes suivantes :

1. un nombre ne doit apparaître qu'une seule fois par ligne,
2. un nombre ne doit apparaître qu'une seule fois par colonne,
3. un nombre ne doit apparaître qu'une seule fois par sous-bloc.

Modéliser une grille de sudoku en CSP revient donc, par exemple, à représenter les ensembles $\mathcal{C}, \mathcal{V}$ et $\boldsymbol{D}$ de la manière suivante :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 4 | | | | | 6 | | | 1 |
| | | | 2 | | | | 6 | |
| 2 | | 5 | | 9 | 7 | | | |
| | 2 | 7 | | | | | | |
| 9 | 1 | | | 4 | | | 2 | 6 |
| | | | | | | 1 | 5 | |
| | | | 4 | 5 | | 8 | | 3 |
| | 3 | | | | 1 | | | |
| 8 | | | 3 | | | | | 2 |

Figure 3.3 – Exemple de grille de sudoku.

**Modélisation du problème** L'ensemble $\mathcal{V} = \{V_{i,j} | \forall i, j \in [1..9]\}$ représente l'ensemble des variables du problème et est constitué de l'ensemble des cases de la grille. La variable $V_{1,1}$ représente la case de la $1^e$ ligne et la $1^e$ colonne, etc. Nous avons donc besoin de 81 variables pour modéliser un sudoku en tant que CSP.

L'ensemble $\boldsymbol{D} = \{D_{i,j} | \forall i, j \in [1, ..9]\}$ représente l'ensemble des valeurs possibles pour chacune des variables de $\mathcal{V}$. Le domaine $D_{1,1}$ correspond aux valeurs possibles pour la case [1,1], *i.e.* la variable $V_{1,1}$. Les cases instanciées voient leur domaine de définition réduit à une seule valeur, le nombre contenu dans la case correspondante. En se basant sur la grille de sudoku présentée en figure 3.3, nous avons par exemple $D_{1,1} = \{4\}$.

L'ensemble $\mathcal{C}$ des contraintes de construction de la grille est décrit par les trois règles énoncées ci-avant. Cet ensemble de relations numériques ou logiques doit donc spécifier qu'un nombre ne doit apparaître qu'une seule fois par ligne et par colonne et une seule fois par sous-carré. L'unicité des chiffres

de 1 à 9 dans un sous-carré peut par exemple être modélisée de la façon suivante :

$$\begin{aligned} V_{1,1} &\neq V_{1,2} \\ V_{1,1} &\neq V_{1,3} \\ &\dots \\ V_{3,2} &\neq V_{3,3} \end{aligned}$$

Cet ensemble de 36 relations suffit uniquement à modéliser le fait que les cases du premier sous-carré en haut à gauche sont toutes différentes deux à deux. Il faut donc renouveler cette modélisation pour les 8 sous-carrés restants, et rajouter les contraintes concernant les lignes et les colonnes.

Afin de simplifier l'écriture, les approches CSP nous permettent d'utiliser des quantificateurs, la modélisation des 36 relations précédentes s'en trouve simplifiée :

$$\forall i, j, k, l \in [1..3], V_{i,j} \neq V_{k,l}, i \leq k, j \leq l$$

Les contraintes présentées ci-dessus sont définies sur un couple de variables, on parle alors de contraintes binaires. La communauté de programmation par contraintes a proposé le développement de contraintes prenant en compte un nombre quelconque de variables : les *contraintes globales*. Un exemple de contrainte globale pouvant être utilisée dans la modélisation de sudoku est la contrainte *alldiff*. Elle permet de spécifier que toutes les variables passées en paramètres doivent avoir des valeurs différentes, nous définissons donc la contrainte suivante pour le premier sous-carré de notre grille de sudoku :

$$\textbf{alldiff}(V_{1,1}, V_{1,2}, V_{1,3}, V_{2,1}, V_{2,2}, V_{2,3}, V_{3,1}, V_{3,2}, V_{3,3})$$

Les contraintes globales mettent en jeu des algorithmes de consistance extrêmement performants et sont en pratique beaucoup plus efficaces que la conjonction des contraintes simples qu'elles peuvent représenter (dans le cas de *alldiff* par exemple). Beldiceanu *et al.* [BCDP07] ont dressé un catalogue (disponible en ligne [www]) des contraintes globales.

La modélisation d'une grille de sudoku en tant que CSP n'étant utilisée ici que pour exemplifier la méthodologie générale de résolution de problèmes sous contraintes, nous ne détaillons pas toutes les contraintes représentant la grille présentée en figure 3.3.

**Résolution** Une fois le problème modélisé, il faut le résoudre. L'avantage des systèmes de programmation par contraintes est d'abstraire l'utilisateur du « *comment* » (*i.e.* la résolution du problème) pour mieux se concentrer sur le « *quoi* » (*i.e.* la modélisation). Nous devons donc uniquement spécifier l'ensemble des contraintes modélisant un problème. Un algorithme de résolution est ensuite utilisé pour calculer l'ensemble des solutions (si elles existent). Les systèmes de résolution de contraintes sont basés sur les notions d'*énumération*, de *filtrage* et de *propagation*.

Une première solution envisagée pour la résolution d'une grille de sudoku est de tenter de donner une valeur à chacune des cases du sudoku, puis de vérifier si les contraintes sont satisfaites : cela correspond à une *énumération* exhaustive. Cette technique présente l'avantage d'être sure de trouver la solution du CSP (ici le sudoku), mais elle présente le désavantage d'être extrêmement coûteuse puisqu'il faut tester pour chacune des cases vides du sudoku toutes les possibilités possibles. Dans l'exemple présenté en figure 3.3, le sudoku comporte $54$ cases vides. Si nous voulons réaliser une énumération exhaustive (c'est-à-dire tester pour chaque case toutes les valeurs possibles sans tenir compte des contraintes),

il nous faut tester $54^9 = 3904305912313344$ possibilités, ce qui à raison d'une possibilité testée par milliseconde prendra approximativement $123804$ ans. Cette énumération exhaustive, également appelée méthode d'« essai-erreur » (*generate and test*), est une recherche systématique qui ne prend pas en compte les contraintes du problème lors de l'affectation des variables, son efficacité est donc déplorable. En effet, le nombre d'affectations testées par cette méthode de recherche complète est de l'ordre de la taille du produit Cartésien des domaines de toutes les variables.

Afin d'améliorer la méthode naïve d'essai-erreur, une autre méthode de recherche exhaustive a été développée : le retour arrière (*backtracking*). Cette méthode consiste toujours en une exploration en profondeur de l'ensemble des possibilités mais, cette fois, les contraintes sont prises en compte lors de la phase d'instanciation des variables. À chaque nouvelle valeur affectée à une variable, nous testons la satisfaction des contraintes. Si l'affectation réalisée aboutit à une violation des contraintes, l'instanciation est invalidée et l'algorithme revient au dernier point de choix consistant (c'est-à-dire qui satisfait les contraintes) afin de tester une autre possibilité.

Dans notre exemple de sudoku, supposons que nous souhaitons instancier la case $V_{1,2}$ à la valeur $6$. En testant la satisfaction des contraintes, nous nous apercevons que cette affectation n'est pas consistante puisque la valeur $6$ apparaît déjà sur la $1^e$ ligne (en case $V_{1,6}$). Ainsi qu'elle ne peut être affectée à la case $V_{1,2}$ et nous évitons ainsi de tester toutes les configurations dans lesquelles $V_{1,2} = 6$. Le mécanisme de retour arrière permet ainsi d'éviter de tester bon nombre d'instanciations ne pouvant amener à une solution, à l'inverse du *generate and test*. Toutefois, pour la plupart des problèmes sa complexité algorithmique reste exponentielle.

Nous venons de présenter les méthodes de recherche systématique pour la résolution de CSP, à savoir le *generate and test* et le *backtracking*. Ces deux approches ne sont pas efficaces car elles ne prennent pas en compte les contraintes lors de l'affectation des variables (*generate and test*) ou bien ne les prennent en compte qu'une fois l'affectation d'une variable faite (*backtracking*), elles explorent donc inutilement de trop nombreuses configurations. Afin de diriger ces recherches en évitant les configurations inconsistantes, nous pouvons mettre en œuvre des techniques de *filtrage* qui vont réduire l'espace de recherche. Pour améliorer l'algorithme de *backtracking* présenté précédemment, nous pouvons essayer de prévoir les conséquences que vont avoir l'instanciation d'une variable sur les variables non encore instanciées du problème, c'est le mécanisme de *look-ahead*. En effet, si lorsque nous affectons une valeur à une variable $V_{i,j}$ du sudoku, nous nous apercevons que le domaine $D_{k,l}$ de la variable $V_{k,l}$ devient vide, c'est que l'affectation courante amène à une inconsistance. Nous devons donc effectuer un retour arrière pour tester une autre valeur pour $V_{i,j}$. Ce principe de filtrage met en œuvre des notions de *consistances locales* qui permettent d'éliminer des domaines des variables les valeurs qui ne peuvent appartenir à aucune solution et ce en fonction d'une affectation partielle courante. La sous-section 3.2.4 du chapitre courant est dédiée à la présentation des principales consistances locales utilisées pour la résolution des problèmes de satisfaction de contraintes numériques.

Enfin, il est possible d'accroître la réduction des espaces de domaines en couplant au filtrage des méthodes de *propagation de contraintes* (cf. section 3.2.5), qui vont tirer profit de l'ensemble des contraintes du problème. Le CSP est alors vu comme un réseau de contraintes (également appelé *store*) qui relie les variables entre elles. La propagation de contraintes consiste, lorsque l'on détecte une réduction d'un domaine d'une variable, à « réveiller » toutes les contraintes dans lesquelles apparaît cette variable. Nous effectuons alors des déductions sur les domaines des autres variables impliquées dans les contraintes réveillées et tentons de réduire leurs domaines. Ce processus est appliqué jusqu'à obtention d'un *point fixe* pour lequel plus aucune contrainte n'est réveillée, plus aucun domaine n'est réduit ou jusqu'à détection d'une inconsistance du CSP, c'est-à-dire que celui-ci n'admet aucune solution.

En reprenant l'analogie avec notre exemple de sudoku présenté en figure 3.3, supposons que nous

venons d'instancier la variable $V_{5,5}$ (en gras sur la figure 3.4) à la valeur $4$. En propageant les contraintes d'unicité d'un chiffre dans une ligne et une colonne, nous pouvons déduire que la ligne $L_5$ et la colonne $C_5$ ne doivent plus contenir de $4$. En poursuivant notre déduction sur les valeurs $4$ instanciées, nous observons que les lignes $L_1$ et $L_7$ ne peuvent contenir de $4$, tout comme les colonnes $C_1$ et $C_4$, les variables $V_{1,1}$ et $V_{7,4}$ étant instanciées à cette valeur. Enfin, nous pouvons réduire le domaine de la variable $V_{2,6}$ à la valeur $4$, puisque le bloc mis en évidence sur la figure 3.4 doit obligatoirement contenir un $4$ (contrainte d'appartenance de chacun des chiffres à un sous-carré) et que la case $[2, 6]$ est la seule disponible concernant cette valeur.

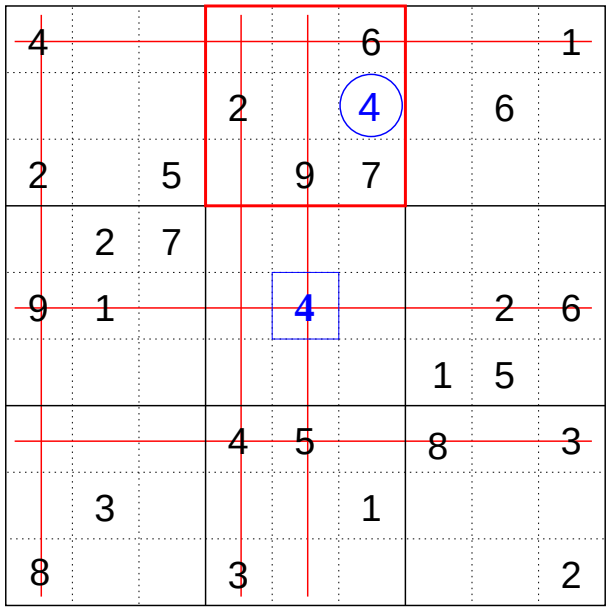



Figure 3.4 – Propagation de contraintes dans un sudoku.

Nous ne détaillons pas ici les méthodes utilisées pour la résolution de sudoku. Le lecteur intéressé par l'application de méthodes de programmation par contraintes à la résolution de sudoku est invité à se référer à l'ouvrage de Narendra Jussien [Jus06]. Toutefois, nous pouvons remarquer que les techniques mises en œuvre sont assez proches de celles intuitivement employées à la main par un humain devant sa grille de sudoku. Il s'agit pour une case de raisonner sur les valeurs possibles (et ainsi d'éliminer celles qui ne le sont pas), d'appliquer les contraintes localement, puis de propager l'information acquise aux régions voisines afin de réitérer le raisonnement. Nous effectuons ces étapes jusqu'à obtention d'un domaine de définition de taille 1 pour chacune des variables du problème.

Nous venons de présenter les concepts et définitions de base d'un CSP, nous allons maintenant nous intéresser en détails à une sous-catégorie de problèmes de satisfaction de contraintes : les CSP numériques.

## 3.2 Les problèmes de satisfaction de contraintes numériques — NCSP (*Numerical CSP*)

Les problèmes de satisfaction de contraintes numériques, ou NCSP, sont l'extension continue des CSP classiques. Ces approches sont basées sur l'utilisation de domaines continus afin de représenter les ensembles de définition des variables, à l'inverse des CSP classiques qui représentent les domaines des variables par des ensembles de valeurs discrètes. Les approches complètes de résolution numérique permettent de s'attaquer à de nombreux problèmes réputés difficiles issus de domaines d'application divers comme la conception préliminaire, la conformation moléculaire, l'automatique, la robotique ou la synthèse d'images par exemple.

Les méthodes de résolution associées calculent un ensemble de pavés (produits Cartésiens multi-dimensionnels) approximant par l'extérieur l'ensemble réel de solutions en assurant qu'aucune d'elles n'est perdue pendant le processus de résolution : c'est la propriété de *complétude*. Cette propriété est assurée par l'utilisation de l'analyse par intervalles dans les techniques de résolution.

### 3.2.1 L'analyse par intervalles

Dans la suite de cette section, nous limitons volontairement la présentation de l'analyse par intervalles aux concepts utilisés dans la suite du chapitre. Pour une présentation complète de l'analyse d'intervalles, le lecteur est invité à se référer aux ouvrages suivants [Moo66, AH83, Neu90].

La résolution de contraintes numériques consiste à effectuer des calculs sur des relations numériques portant sur des variables définies en virgule flottante (les seuls chiffres représentables en machine). Ce processus peut être considéré comme trivial, mais il ne l'est pas. Cela est dû aux limitations dont souffre la représentation des chiffres réels par des nombres flottants dans les ordinateurs. Prenons un exemple simple illustrant bien ces limitations.

**Exemple 3.2 (Limitations dues à la représentation en nombres flottants).** *Soit $f(x, y)$ la fonction de Rump [Rum88] définie comme suit:*

$$f(x,y) = 333,75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5,5y^8 + \frac{x}{2y}$$

*Le calcul de $f(77617, 33096)$ sous MuPAD 1.4.2 et avec un processeur AMD Athlon donne les résultats suivants :*

$$\begin{cases} \textit{7 bits de précision} : & -2.47588e^{27} \\ \textit{16 bits de précision} : & -5.7646075226366 51e^{17} \\ \textit{24 bits de précision} : & -134217728.0 \end{cases}$$

*Pourtant, le calcul exact de cette instanciation est :*

$$f(77617, 33096) = -\frac{54767}{66192} \approx -0.82739605$$

Cet exemple montre bien les limites de la représentation informatique finie des nombres réels. Les ordinateurs étant par nature constitués d'éléments de nature finie, ils ne peuvent stocker et manipuler qu'un nombre fini de nombres réels. Ce sous-ensemble représentable noté $\mathbb{F}$, est constitué des nombres dits à *virgule flottante*, et sa définition est régie par la norme IEEE 754 [IEE85].

#### *3.2.1.1 La norme IEEE 754*

La norme IEEE 754 spécifie les règles de représentation des nombres flottants en machine, un nombre réel $x$ est définit par :

1. un bit de signe $S$,
2. un exposant $E$,
3. une mantisse (ou significande) $M$,
4. une base $B$ (2 dans le cas des ordinateurs qui utilisent le binaire comme base de référence),

chacune de ces données (à l'exception de $B$) est codée sur un nombre de bits dépendant de la précision recherchée et donc du type de la variable informatique représentée.

Trois nombres spécifiques sont également introduits par cette norme :

- $+\infty$, représente la borne supérieure absorbante des calculs,
- $-\infty$, la borne absorbante inférieure,
- NaN (*Not a Number*), permet de représenter les résultats d'opérations invalides ($0/0$ par exemple).

Pour une présentation plus détaillée de la norme IEEE 754, de la représentation des réels en machine et des concepts associés, le lecteur est invité à consulter la thèse de Frédéric Goualard [Gou00].

L'ensemble $\mathbb{F}$ des nombres représentables en machine, appelés nombres flottants ou plus simplement *flottants*, générés à partir de cette norme n'est qu'un sous-ensemble très restreint des nombres réels. De plus, $\mathbb{F}$ n'est pas un ensemble fermé pour les opération arithmétiques, cela signifie que des opérations d'arrondis permettant de passer de $\mathbb{R}$ à $\mathbb{F}$ doivent être mises en place. La norme IEEE 754 propose quatre formes d'arrondis différentes pour un réel $x$ :

1. l'arrondi au flottant le plus proche ($[x]_{\mathbb{F}}$),
2. l'arrondi vers zéro ($[x]_0$),
3. l'arrondi vers $+\infty$ ($\lceil x \rceil$),
4. l'arrondi vers $-\infty$ ($\lfloor x \rfloor$).

La figure 3.5 illustre les différents modes d'arrondis proposés par la norme IEEE 754. Sur ce schéma, étant donné un nombre flottant $f$, nous représentons le plus petit flottant immédiatement supérieur (resp. le plus grand flottant immédiatement inférieur) à $f$ par $f^+$ (resp. $f^-$).

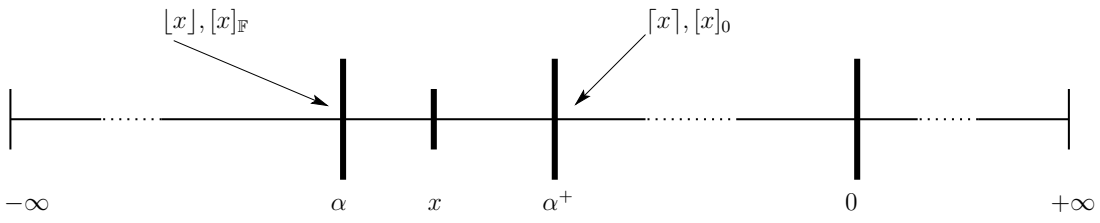



Figure 3.5 – Les différents modes d'arrondis d'un nombre réel $x$ définis par la norme IEEE 754, d'après Goualard [Gou00].

Nous disons d'une opération qu'elle est *correctement arrondie* lorsque son résultat, si il n'est pas représentable en machine, est le flottant le plus proche. Il est à noter que dans le standard IEEE 754, tous les opérateurs ne sont pas correctement arrondis, les seuls jouissant de cette propriété sont les

opérateurs : $+, -, \times, \div, \sqrt{}$, la précision des autres opérateurs est dépendante de l'implémentation. Le lecteur intéressé est invité à se référer au papier de Lefèvre *et al.* [LMT98] sur ce sujet. Cette notion d'arrondi rend particulièrement difficile les implémentations sûres de systèmes d'équations et d'inéquations non-linéaires et ce sujet de recherche a donné lieu à une littérature particulièrement abondante [Knu81, Gol91, LM01]. Une des solutions permettant de mieux contrôler les problèmes d'arrondis dûs à l'arithmétique en nombre flottant est de recourir à l'*arithmétique des intervalles*, que nous présentons dans la section suivante.

## 3.2.2 L'arithmétique des intervalles

L'arithmétique des intervalles a été proposée par Moore [Moo66] et les concepts sous-jacents ont été développés dans la thèse de Frédéric Goualard [Gou00]. Cette solution consiste à remplacer l'utilisation des nombres réels ($\in \mathbb{R}$) par des *intervalles à bornes flottantes* ($\in \mathbb{F}$ et donc représentables en machine), les contenant. Cette idée est à rapprocher de la représentation de nombres transcendantaux, à l'instar de $\pi$ que nous pouvons représenter par l'intervalle de réels $[3,14; 3,15]$. Une configuration $n$-dimensionnelle est quant à elle représentée par un produit Cartésien de $n$ intervalles, appelé *boîte* ou *pavé*.

### *3.2.2.1 Définitions*

**Définition 3.5** (Intervalle à bornes flottantes)**.** Un intervalle $I$ est dit à bornes flottantes si il est de la forme :

$$I = [a, b] = \{r \in \mathbb{R} \,|a \leq r \leq b, \text{ avec } a, b \in \mathbb{F}\}$$

**Définition 3.6** (Intervalle canonique à bornes flottantes)**.** Un intervalle $I$ non-vide à bornes flottantes de la forme :

$$I = [a, b] \text{ tel que } b \leq a^{+}, \text{ avec } a, b \in \mathbb{F}$$

est dit *canonique*.

**Définition 3.7** (Pavé canonique)**.** Un produit Cartésien d'intervalles à bornes flottantes (*i.e.* une boîte ou un pavé) est dit *canonique* s'il l'est dans chacune de ses dimensions, c'est-à-dire si chacun des intervalles du produit Cartésien est canonique.

Dans la suite de ce chapitre, nous adoptons les conventions d'écriture suivantes :
- les vecteurs ou les produits Cartésiens (boîtes) sont en **gras**,
- $\mathbb{I}_{\mathbb{F}}$ (ou $\mathbb{I}$ par défaut) représente l'ensemble des intervalles à bornes flottants,
- $\mathbb{I}_{\mathbb{R}}$ représente l'ensemble des intervalles à bornes réelles,
- le terme *intervalle* fait référence, sauf précision, aux intervalles à bornes flottantes.

Dans nos travaux, nous n'utilisons que la notion d'intervalle *fermé*, pour plus d'informations sur les notions d'intervalles ouverts et fermés, le lecteur est invité à consulter [AH83, Gou00]. Afin d'assurer la clôture de l'ensemble $\mathbb{I}_{\mathbb{R}}$, il est indispensable de redéfinir les opérateurs usuels. Moore [Moo66] propose de définir une *extension aux intervalles* des opérations de base. Cette extension jouit de la propriété d'inclusion (*containment*).

### *3.2.2.2 Extension aux intervalles*

**Définition 3.8** (Extension aux intervalles)**.** Soit $\mathbf{B} \in \mathbb{I}^n$ une boîte de dimension $n$ et $f$ une fonction sur les réels $f : \mathbb{R}^n \to \mathbb{R}$. Soit l'application de $f$ à $\mathbf{B}$ définie comme suit : $f(\mathbf{B}) = \{f(r)|r \in \mathbb{F}^n, r \in \mathbf{B}\}$. Enfin, soit $\mathcal{D}_f$ le domaine de définition de $f$ et $\mathcal{D}_f^{\mathbb{I}} = \{\mathbf{B} \in \mathbb{I}^n | \mathbf{B} \subseteq \mathcal{D}_f\}$.

Une définition de l'*extension aux intervalles* proposée par Neumaier [Neu90] consiste en une fonction $F : \mathbb{I}^n \rightarrow \mathbb{I}$, extension de $f$, définie sur les intervalles et vérifiant les propriétés suivantes :

$$f(r) = F(r), \forall r \in \mathcal{D}_f \quad (3.1)$$
$$\square(f(\mathbf{B})) \subseteq F(\mathbf{B}), \forall \mathbf{B} \in \mathcal{D}_f^{\mathbb{I}} \quad (3.2)$$

où $\square(\rho) = \bigcap\{\mathbf{B} \in \mathbb{I}^n | \rho \subseteq \mathbf{B}\}$ représente pour toute relation réelle, $\rho \in \mathbb{R}^n$, sa plus petite boîte englobante, aussi appelé opérateur Hull.

Notons que si la fonction ne vérifie que l'équation 3.2, l'extension est alors appelée une *extension faible aux intervalles*.

Une extension aux intervalles $\mathbb{I}_{\mathbb{R}}$ générique pour les opérateurs sur les réels peut être définie comme suit :

$$[a,b] \Diamond [c,d] = \square\{x \diamond y | x \in [a,b], y \ \in [c,d]\}$$

Cette définition générique est instanciée de la manière suivante aux intervalles de réels ($\mathbb{I}_{\mathbb{R}}$) :

$$\begin{aligned}
[a,b] + [c,d] &= [a+c, b+d] \\
[a,b] - [c,d] &= [a-d, b-c] \\
[a,b] \times [c,d] &= [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \\
[a,b] \div [c,d] &= [\min(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d}), \max(\frac{a}{c},\frac{a}{d},\frac{b}{c},\frac{b}{d})] \\
\exp([a,b]) &= [\exp(a), \exp(b)]
\end{aligned}$$

Afin de clôturer l'ensemble $\mathbb{I}_{\mathbb{F}}$, nous devons arrondir correctement ces opérations sur les flottants, c'est-à-dire implémenter les extensions présentées ci-dessus de la manière suivante :

$$\begin{aligned}
[a,b] + [c,d] &= [\lfloor a+c \rfloor, \lceil b+d \rceil] \\
[a,b] - [c,d] &= [\lfloor a-d \rfloor, \lceil b-c \rceil] \\
[a,b] \times [c,d] &= [\min(\lfloor ac \rfloor, \lfloor ad \rfloor, \lfloor bc \rfloor, \lfloor bd \rfloor), \max(\lceil ac \rceil, \lceil ad \rceil, \lceil bc \rceil, \lceil bd \rceil)] \\
[a,b] \div [c,d] &= [\min(\lfloor \tfrac{a}{c} \rfloor, \lfloor \tfrac{a}{d} \rfloor, \lfloor \tfrac{b}{c} \rfloor, \lfloor \tfrac{b}{d} \rfloor), \max(\lceil \tfrac{a}{c} \rceil, \lceil \tfrac{a}{d} \rceil, \lceil \tfrac{b}{c} \rceil, \lceil \tfrac{b}{d} \rceil)] \\
\exp([a,b]) &= [\lfloor \exp(a) \rfloor, \lceil \exp(b) \rceil]
\end{aligned}$$

Les extensions aux intervalles sont définies en pratique comme des fonctions monotones par rapport à l'inclusion.

**Définition 3.9** (Fonction monotone par rapport à l'inclusion)**.** Une fonction étendue aux intervalles $F : \mathbb{I}^n \rightarrow \mathbb{I}$ est dite *monotone par rapport à l'inclusion* si elle vérifie la propriété suivante :

$$\forall \mathbf{B}, \mathbf{D} \in \mathbb{I}^n, \mathbf{B} \subseteq \mathbf{D} \Rightarrow F(\mathbf{B}) \subseteq F(\mathbf{D})$$

À partir de cette définition, Moore [Moo66] définit le théorème clé de l'arithmétique des intervalles :

**Théorème 3.1.** *Soit un intervalle $I \in \mathbb{I}^n$, une fonction réelle $f$ et son extension $F$ aux intervalles, monotone par rapport à l'inclusion. Nous avons alors :*

$$\forall x \in I \Rightarrow f(x) \in F(I)$$

Nous définissons alors une extension *naturelle* aux intervalles d'une fonction réelle $f$ comme étant la fonction $F$ dans laquelle toute variable réelle a été remplacée par son équivalent intervalle et tout opérateur définit sur les réels a été remplacé par son extension aux intervalles.

**Exemple 3.3 (Extension naturelle aux intervalles).** *Soit la fonction $f$ suivante définie sur les réels :*

$$f(x, y) = x^2 - (x \times y) + 2 | x, y \in \mathbb{R}$$

*L'extension naturelle $F$ de la fonction $f$ est alors définie comme suit:*

$$F(X, Y) = X^2 \ominus (X \otimes Y) \oplus [2, 2] | \; X, Y \in \mathbb{I}$$

*où $\ominus, \otimes$ et $\oplus$ représentent respectivement les extensions aux intervalles des opérateurs $-, \times$ et $+$.*

Toutefois, il faut noter que la substitution des opérateurs réels par leur extension aux intervalles induit une surévaluation de la fonction. Cette perte de précision peut être limitée par application du théorème des *occurrences simples* introduit par Moore [Moo66] :

**Théorème 3.2** (Théorème des occurrences simples [Moo66])**.** *Soit une fonction réelle $f : \mathbb{R}^n \to \mathbb{R}$ telle qu'aucune de ses variables n'apparaissent plus d'une fois, et soit $F$ l'extension naturelle aux intervalles de $f$. Nous avons alors, aux erreurs d'arrondis près :*

$$F(X_1, \ldots, X_n) = \{ f(x_1, \ldots, x_n) | x_i \; \in X_i, \forall i \in [1, n] \; \}$$

#### *3.2.2.3 Propriétés des opérateurs de l'arithmétique des intervalles*

Les opérateurs arithmétiques définis sur les intervalles jouissent des propriétés suivantes, $\forall X, Y, Z \in \mathbb{I}_{\mathbb{R}}$ :

| | | | | |
|---|---|---|---|---|
| $(X + Y)$ | $= (Y + X)$ | $XY$ | $= YX$ | commutativité |
| $(X + Y) + Z$ | $= X + (Y + Z)$ | $(XY)Z$ | $= X(YZ)$ | associativité |
| $X(Y + Z)$ | $\subseteq XY + XZ$ | | | sous-distributivité |

Les opérations définies sur $\mathbb{I}_{\mathbb{F}}$ ne vérifient pas la propriété d'associativité, ainsi des formes équivalentes de fonctions spécifiées sur les réels peuvent ne pas avoir la même extension naturelle.

Nous venons de présenter succinctement les concepts de l'arithmétique des intervalles et des extensions liées à la gestion correcte des nombres flottants en machine. Nous allons maintenant présenter l'application des notions d'intervalles à des contraintes numériques.

### 3.2.3 Les contraintes d'intervalles

Comme nous l'avons présenté en introduction, l'arithmétique des intervalles est une solution qui permet de prendre en compte les problèmes d'arrondis inhérents à la résolution de contraintes numériques. Les méthodes classiques de résolution de relations (équations ou inéquations, linéaires ou non) peuvent être adaptées à l'arithmétique des intervalles. Ainsi, il existe des versions intervalles des méthodes de Newton-Raphson ou de Gauss-Seidel permettant la résolution de contraintes d'intervalles. De même, des opérateurs spécifiques (Krawczyk ou Hansen-Segupta par exemple) ont été étendus aux intervalles, permettant ainsi leur utilisation.

Les extensions intervalles de ces algorithmes numériques présentent un certain nombre de propriétés avantageuses :

- la complétude,

- la preuve d'existence de solutions (voire même d'unicité dans certains cas),
- l'assurance de convergence pour les algorithmes itératifs modulo l'existence d'un point-fixe dans le domaine d'entrée.

Mais elles souffrent également de certains inconvénients :

- leur efficacité est généralement faible,
- la notion de préférence dans la prise en compte des contraintes (ces techniques sont plus efficaces sur certains types de contraintes que sur d'autres),
- la coopération des techniques n'est pas aisée à mettre en œuvre.

### 3.2.4 Consistances locales

Les notions de consistances locales sont issues de l'intelligence artificielle, on y trouve en particulier la notion d'*arc consistance* [Mac77, Bes94]. Un filtrage par consistance d'arc consiste à éliminer les éléments des domaines des variables qui ne peuvent trivialement appartenir à aucune solution, c'est-à-dire que l'on supprime de chaque domaine toutes les valeurs sans *support*. Une valeur $d_i$ d'un domaine $D_i$ a un support $d_j$ dans le domaine $D_j$ si le couple $(d_i, d_j)$ est autorisé par les contraintes liant les variables $V_i$ et $V_j$. Appliquées à la résolution de problèmes de satisfaction de contraintes, elles consistent en l'élimination des domaines de définition des variables (*i.e.* des pavés) des valeurs ne permettant la satisfaction des contraintes du problème. La consistance d'arc (ou arc-consistance) ne pouvant être obtenue pour des nombres flottants, la communauté de programmation par contraintes a défini de nouvelles consistances locales, approximations de la consistance d'arc.

La définition d'une consistance correspond en l'approximation d'une relation réelle $\rho$ par une boîte en assurant de ne perdre aucune des solutions de $\rho$. L'élimination des valeurs inconsistantes se fait par application d'un opérateur d'approximation extérieure (ou opérateur de contraction), qui doit vérifier les propriétés de *contractance*, *monotonie* et de *correction*. Nous pouvons bien évidemment définir plusieurs approximations extérieures pour une même contrainte en fonction de la consistance prise en compte. Nous proposons dans la suite une définition d'un opérateur d'approximation extérieure, avant de présenter les deux principales consistances locales spécifiques aux problèmes de satisfaction de contraintes : la *hull-consistance* [Ben95] et la *box-consistance* [BMvH94].

#### *3.2.4.1 Approximations extérieures*

Éliminer toutes les valeurs inconsistantes d'une boîte est un problème insoluble pour un système de contraintes réelles. Les méthodes de consistance locales reposent donc sur l'application d'opérateurs de contractance extérieure qui éliminent des valeurs d'une boîte en fonction d'une consistance donnée.

**Définition 3.10** (Opérateur d'approximation extérieure)**.** Soit $c$ une contrainte $n$-aire, et $\omega$ une consistance donnée. Un opérateur d'approximation extérieure pour $c$ est une fonction $\mathsf{Outer}_c^\omega : \mathbb{I}^n \rightarrow \mathbb{I}$ qui élimine des valeurs de $\mathbb{I}^n$ en fonction de $\omega$.

Nous présentons maintenant deux consistances locales utilisées en programmation par contraintes, la figure 3.6 illustre les différences entre la consistance d'arc, la *hull*-consistance et la *box*-consistance.

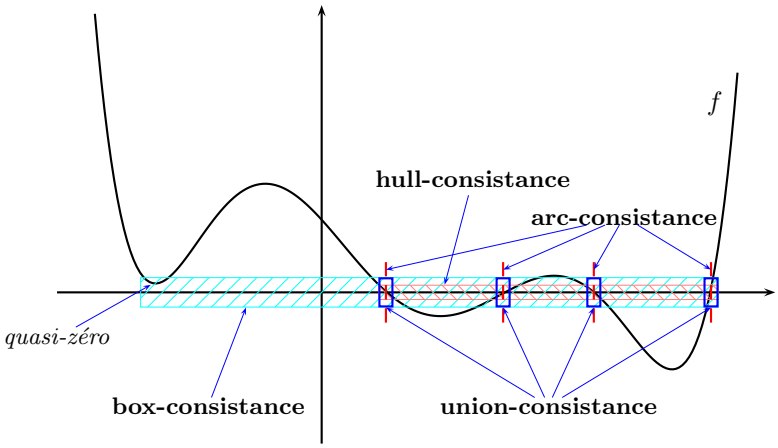



Figure 3.6 – Illustration de différentes consistances locales, d'après Goualard [Gou00].

#### *3.2.4.2* Hull-*consistance*

**Définition 3.11** (Hull-consistance)**.** Soit $c(x_1, \ldots, x_n)$ une contrainte réelle et $\rho_c$ la relation associée. La contrainte $c(x_1, \ldots, x_n)$ est dite *hull-consistante par rapport au pavé* $I$ ssi $\forall i \in \{1, \ldots, n\}$ :

$$I_i = \mathsf{Hull}\big(I_i \cap \big\{a_i \in \mathbb{R} \mid \forall j \in \{1, \ldots, n\} \setminus \{i\} : \exists a_j \in I_j \text{ t.q. } (a_1, \ldots, a_n) \in \rho_c\big\}\big)$$

#### *3.2.4.3* Box-*consistance*

**Définition 3.12** (Box-consistance)**.** Soit $c$ une contrainte $n$-aire sur les réels, $C$ une extension aux intervalles de $c$, et $\boldsymbol{B} = I_1 \times \cdots \times I_n$ un pavé. La contrainte $c$ est *box-consistante* par rapport à $\boldsymbol{B}$ ssi $\forall k \in \{1, \ldots, n\}$ :

$$I_k \quad = \quad \mathsf{Outer}(I_k \cap \{r \in \mathbb{R} \mid C(I_1, \ldots, I_{k-1}, \mathsf{Outer}(\{r\}), I_{k+1}, \ldots, I_n)\})$$

### 3.2.5 Propagation de contraintes

La propagation de contraintes a pour but d'accélérer le processus de résolution d'un CSP, en réduisant les domaines de recherche des variables du problème c'est-à-dire en éliminant des valeurs inconsistantes de l'espace de recherche initial. L'algorithme général de propagation de contraintes consiste à appliquer à chacune des contraintes $C_i$ du CSP $\mathcal{S} = \langle \mathcal{C}, \mathcal{V}, \boldsymbol{D} \rangle$, un opérateur de réduction de domaine (ou opérateur de *narrowing*) afin d'éliminer les valeurs inconsistantes des domaines des variables. Dans le cadre des CSP numériques, qui nous intéresse dans cette thèse, l'application de cet opérateur de réduction des domaines consiste à modifier les bornes des intervalles $D_i$ pour chaque variable $V_i$ de la contrainte $C_i$.

Le principe général de l'algorithme de propagation de contraintes numériques est énoncé par Frédéric Benhamou [Ben95] :

1. une contrainte $C_i \in \mathcal{C}$ est choisie,
2. l'opérateur de *narrowing* $N_i^c$ associé à cette contrainte est appliqué et modifie (potentiellement) les domaines $D_j$ des variables $V_j$ impliquées dans $C_i$,

3. si au moins une variable $V_j$ a vu son domaine $D_j$ modifié à l'étape précédente, nous rajoutons à l'ensemble des contraintes à traiter toutes celles contenant une occurrence de $V_j$ afin de réappliquer l'opérateur de réduction associé.
4. nous retournons à la première étape jusqu'à ce que toutes les contraintes aient été traitées ou bien jusqu'à ce qu'aucun domaine ne soit modifié. Nous avons alors atteint un *point-fixe*. Si un des domaines est réduit au domaine vide alors le problème est inconsistant.

---

ALG. 3.1 – L'algorithme de réduction d'intervalles.

```
1  Nar(in: {(c_1, N_1), ..., (c_m, N_m)};
       inout: B ∈ 𝕀^n)
2  begin
3   C ← {c_1, ..., c_m}
4   while ((C ≠ ∅) ∧ (B ≠ ∅))
      % Choisir une contrainte c_i dans l'ensemble C
5     c ← choisirContrainte(C)
6     B' ← N_i^c(B)
7     if (B' ≠ B) then
8       C ← C ∪ {c_j | ∃x_k ∈ var(c_j) ∧ I'_k ≠ I_k}
9       B ← B'
10    end
11    C ← C \ {c}
12  endwhile
13 end
```

---

Une fois ce point-fixe atteint, la résolution du CSP numérique se poursuit par application successive de l'algorithme 3.1 de réduction d'intervalles, et d'un algorithme de bissection (bissection) des domaines des variables. Le choix du domaine à découper dépend d'une heuristique propre au problème étudié et peut être par exemple de choisir toujours le plus grand domaine ou bien encore de choisir les variables tour à tour (selon un principe de *round-robin*). L'algorithme bissection est paramétré également par une finesse de découpe ($\varepsilon$), c'est-à-dire une taille minimale des domaines. Nous arrêtons la bissection des domaines s'ils sont tous d'une taille inférieure à $\varepsilon$. L'algorithme classique de propagation de contraintes : propagation est présenté en table 3.2.

Les techniques de propagation de contraintes permettent de résoudre plus efficacement les CSP en tirant profit des contraintes afin de réduire les domaines des variables et ce jusqu'à détection d'une inconsistance (un des domaines devient vide) ou jusqu'à obtention d'une approximation extérieure de l'ensemble des solutions du CSP étudié.

### 3.2.6 Approximation intérieure d'une relation

Les opérateurs d'approximation extérieure vérifient la propriété de *complétude*, c'est-à-dire qu'il n'élimineront pas de valeurs d'une boîte $\mathbf{B}$ satisfaisant la contrainte $c$ à laquelle ils ont été appliqués.

ALG. 3.2 – L'algorithme de résolution d'un système de contraintes.

```
1 propagation(in: S = {(c_1, N_1), ..., (c_m, N_m)}, B ∈ I^n;
                out: U_o ∈ P(I^n))
1 begin
2   B' ← Nar(S, B)
3   if (non conditionArrêt(B')) then
4     (B_1, B_2) ← bissection(B')
5     U_o ← propagation(S, B_1) ∪ propagation(S, B_2)
6   else
7     U_o ← {B'}
8   end
9 end
```

Il est également possible de définir des opérateurs vérifiant la propriété de *correction* sur une relation $\rho$. Ces opérateurs vont fournir une approximation *intérieure* des relations pour lesquelles il est possible de définir une notion d'intérieur et d'extérieur (*e.g.* les inégalités, cf. figure 3.1). En conséquence, les boîtes résultantes de l'application de ces opérateurs jouissent elles aussi de la propriété de correction, c'est-à-dire que tout point de ces boîtes vérifie la relation $\rho$ et est donc solution du système de contraintes. Nous considérons la définition suivante pour un opérateur d'approximation intérieure :

**Définition 3.13** (Opérateur d'approximation intérieure)**.** Soit $\rho$ une relation $n$-aire définie sur les réels, $\rho \subseteq \mathbb{R}^n$. Un opérateur d'approximation intérieure $\mathsf{Inner} : \mathbb{R}^n \rightarrow \mathbb{R}$ est définit par :

$$\mathsf{Inner}(\rho) = \{\boldsymbol{r} \in \mathbb{R}^n | \mathsf{Outer}(\{\boldsymbol{r}\}) \subseteq \rho\}$$

Une approximation intérieure définie comme telle contient tous les éléments dont la plus petite boîte englobante appartient à la relation. De la même manière, nous pouvons définir l'approximation intérieure d'un ensemble de contraintes, c'est-à-dire d'un CSP numérique.

### 3.2.7 Approximation intérieure d'un CSP

Le calcul de l'approximation intérieure d'un problème CSP consiste à appliquer un opérateur d'approximation intérieure à chacune des contraintes du problème de manière séquentielle. Pour un CSP $\mathcal{S} = \langle \mathcal{C}, \mathcal{V}, \mathbf{D} \rangle$, l'approximation intérieure correspondante est obtenue par application successive de l'opérateur $\mathsf{inner}$ d'approximation intérieure présenté dans la section précédente pour chacune des contraintes $c_i$ du problème, *i.e.* $\forall c_i \in \mathcal{C}$.

### 3.2.8 Opérateur d'extension intérieure

Nous proposons la notion d'*opérateur d'extension intérieure* qui consiste, étant donné une relation $\rho$ et un point $s \subseteq \rho$ solution de cette relation, à étendre $s$ de telle façon que le résultat de l'application de

l'opérateur soit correct. On cherche donc, à partir d'un point solution d'une contrainte, à étendre la solution, c'est-à-dire à augmenter la taille des domaines de la boîte $s$. Collavizza *et al.* [CDR99] proposent une approche permettant de calculer, sur une dimension, une extension des domaines consistants d'une solution correcte d'un NCSP en utilisant des fonctions extremum univariées afin d'évaluer les bornes consistantes minimum et maximum de la relation.

Nous proposons une approche similaire, basée sur la définition suivante d'opérateur d'extension intérieure :

**Définition 3.14** (Opérateur d'extension intérieure)**.** Soit $c$ une contrainte $n$-aire, et $\rho_c$ sa relation réelle sous-jacente. Un opérateur d'extension intérieure pour $c$ est une fonction $\mathsf{OEI}_c : \mathbb{I}^n \rightarrow \mathbb{I}$, vérifiant pour toute boîte $\mathbf{B} \subseteq \rho_c$ :

$$\mathsf{OEI}_c(\mathbf{B}) \subseteq \mathbf{B} \subseteq \rho_c$$

Nous introduisons également la notion d'opérateur optimal d'extension intérieure, déterminant la boîte maximale (au sens de l'inclusion) appartenant à la relation $\rho_c$ :

**Définition 3.15** (Opérateur Optimal d'extension intérieure)**.** Soit $c$ une contrainte $n$-aire, et $\rho_c$ sa relation réelle sous-jacente. Un opérateur optimal d'extension intérieure pour $c$ est une fonction $\mathsf{OptOEI}_c : \mathbb{I}^n \rightarrow \mathbb{I}$, vérifiant pour toute boîte $\mathbf{B} \subseteq \rho_c$ :

$$\nexists \mathbf{B}' \in \mathbb{I}^n | (\mathsf{OptOEI}_c(\mathbf{B}) \supset \mathbf{B}') \wedge (\mathbf{B}' \supseteq \rho_c)$$

L'implémentation de cet opérateur optimal se révèle impossible en pratique, à cause des erreurs d'arrondis commises par la représentation flottante (et l'arithmétique associée) des réels en machine. Nous proposons donc en algorithme 3.3 une implémentation approchée de l'opérateur $\mathsf{OptOEI}$. Ce dernier est paramétré par une contrainte $c$, un domaine de recherche initial $\mathbf{B}$, un pavé consistant $\mathbf{P} \subseteq \rho_c$ et une précision $\varepsilon$.

Une implémentation triviale de l'opérateur $\mathsf{OptOEI}$ consiste à étendre la boîte $\mathbf{P}$ d'une valeur $\varepsilon$ dans chacune de ses dimensions jusqu'à ce qu'elle ne satisfasse plus la contrainte $c$ (ou jusqu'à sortir de l'espace de recherche $\mathbf{B}$) et à retourner l'avant-dernière boîte calculée. Nous proposons ici une implémentation plus efficace basée sur une extension dichotomique en deux étapes :

1. une première itérative permet d'étendre la boîte consistante $\mathbf{P}$ selon la moitié du domaine disponible entre $\mathbf{B}$ et $\mathbf{P}$ pour chacune des variables de la contrainte $c$, jusqu'à obtenir une inconsistance partielle.
2. une deuxième étape réduit de façon similaire les domaines des variables inconsistantes de $\mathbf{P}$ jusqu'à ce qu'elles vérifient à nouveau la relation.

Le processus continue ensuite de manière récursive jusqu'à obtention de la précision désirée pour le plus grand domaine de variable de $\mathbf{P}$.

L'algorithme 3.3 utilise les notations suivantes :

- $\boldsymbol{P}|_{v_i}$ représente le domaine de la variable $v_i$ dans la boîte $\mathbf{P}$,
- $\lfloor I$ représente la borne inférieure de l'intervalle $I$,
- $\lceil I$ représente la borne supérieure de l'intervalle $I$.

L'approche peut être vue comme une extension $n$-dimensionnelle de l'implémentation de l'opérateur de box-consistance [VMD97], qui au lieu de garantir la complétude, garantit la correction des résultats de cet opérateur par rapport à une relation numérique.

La figure 3.7 présente l'extension d'une boîte consistante $\mathbf{P}$ dans un espace de recherche $\mathbf{B}$, en fonction d'une contrainte $c$.

ALG. 3.3 – Algorithme (OEI) d'extension intérieure d'un pavé consistant $\mathbf{P}$ par rapport à une contrainte $c$, dans un espace de recherche $\mathbf{B}$.

```
1  OEI(in: c ∈ C, P ∈ 𝕀^n, B ∈ 𝕀^n, ε ∈ ℝ;
       out: K ∈ 𝕀^n)
2  begin
3   if (plusPetiteDifférence(B, P) ≤ ε) then
4     return P
5   else
      % Boucle d'extension de P
6     while (CS(P, c))
7       P' ← P
8       for each v_i ∈ var(c) do
9             P|_{v_i} ← [ (⌊B⌋_{v_i} + ⌊P⌋_{v_i}) / 2 , (⌈B⌉_{v_i} + ⌈P⌉_{v_i}) / 2 ]
10      endforeach
11    end
      % Boucle de réduction de P
12    while (non CS(P, c))
13      P'' ← P
14      for each v_i ∈ var(c) do
15            P|_{v_i} ← [ (⌊P⌋_{v_i} + ⌊P'⌋_{v_i}) / 2 , (⌈P⌉_{v_i} + ⌈P'⌉_{v_i}) / 2 ]
15      endforeach
16    end
      % Récursion
17    return OEI(P, P'', ε)
18  end
19 end
```

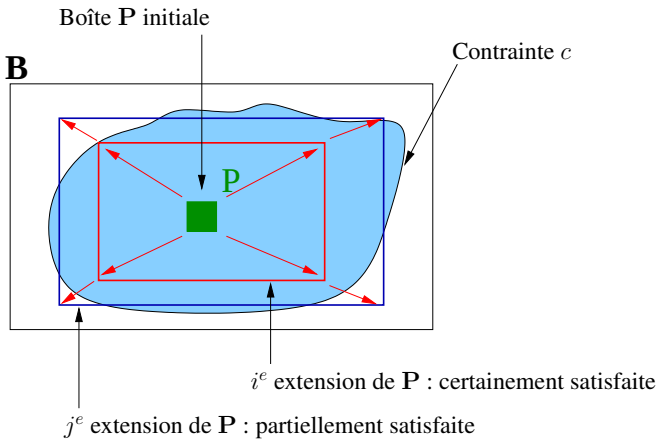



Figure 3.7 – Illustration de l'opérateur OEI (cf. table 3.3) d'extension intérieure appliqué à une contrainte bidimensionnelle.

### 3.2.9 Calcul de l'extension intérieure d'un ensemble de contraintes

Nous venons de présenter l'opérateur permettant de réaliser l'extension intérieure pour une contrainte $n$-dimensionnelle $c$. Nous considérons maintenant une extension intérieure réalisée sur un ensemble de contraintes $\{c_1, \ldots, c_n\}$. Comme précédemment, cette extension s'effectue à l'intérieur d'un pavé de recherche $\mathbf{B}$ et à partir d'une boîte initialement consistante $\mathbf{P}$. Le principe de base est identique à celui présenté dans [CDR99] : nous déterminons l'extension intérieure de chacune des contraintes $c_i$ grâce à l'opérateur OEI, puis nous intersectons les pavés obtenus.

L'algorithme 3.3 présente notre méthode AEI de calcul d'extension intérieure. Ce dernier est paramétré par un ensemble $C = \{c_1, \ldots, c_n\}$ de contraintes numériques $n$-dimensionnelles, une boîte $\mathbf{B}$ représentant l'espace de recherche initial pour le calcul de l'extension intérieure et d'un pavé $\mathbf{P}$ initialement consistant.

ALG. 3.4 – Algorithme AEI d'extension intérieure pour un CSP numérique, *i.e.* un ensemble de $n$ contraintes numériques $C = \{c_1, \ldots, c_n\}$.

```
1 AEI(in: C, P ∈ 𝕀^n, B ∈ 𝕀^n, ε ∈ ℝ;
        out: K ∈ 𝕀^n)
2 begin
3   return ⋂_{c_i∈C}^n OEI(c_i, P, B, ε)
4 end
```

### 3.2.10 La résolution d'un problème MAX-NCSP

Comme nous l'avons mis en évidence dans le chapitre 2 consacré à l'état de l'art, la description d'un problème de placement de caméra en tant que liste de propriétés (sur les paramètres de la caméra ou bien comme ensemble de configurations visuelles à obtenir) peut amener à la spécification d'un problème sur-contraint. L'introduction de cette section a quant à elle mis en évidence le fait que les solutions existantes (contraintes hiérarchiques, méthodes d'optimisation ou de résolution de contraintes classiques) ne sont pas suffisantes pour prendre en compte correctement la résolution de ces problèmes. C'est pourquoi nous proposons une méthode de résolution de problème de placement de caméra sous contraintes en tant que résolution de problème MAX-NCSP.

Les propriétés utilisateur sont représentées chacune comme une conjonction de contraintes ayant un lien sémantique, le problème sera donc composé d'un ensemble de ces ensembles « *sémantiques* » de contraintes. Une fois le problème ainsi spécifié, l'objectif est de déterminer une approximation intérieure de cet ensemble de contraintes satisfaisant le maximum de propriétés possibles (*i.e.* d'ensembles sémantiques de contraintes). À l'issue du processus de résolution, nous sommes en mesure de fournir à l'utilisateur : soit (*i*) une solution au problème initial (si celui-ci est bien-contraint), soit (*ii*) les meilleures solutions possibles en termes de propriétés satisfaites, ainsi que pour chacune de ces solutions partielles la liste des propriétés satisfaites (ou non).

#### *3.2.10.1 Définitions*

Avant de détailler l'approche de résolution MAX-NCSP, nous proposons tout d'abord quelques définitions.

**Définition 3.16** (Problème MAX-NCSP)**.** Un problème MAX-NCSP est un problème potentiellement sur-contraint, défini par $\mathcal{S} = \langle C, \mathcal{V}, \boldsymbol{D} \rangle$, avec $C$ un ensemble de contraintes numériques, $\mathcal{V}$ un ensemble de variables et $\boldsymbol{D}$ un produit Cartésien d'intervalles à bornes flottantes représentant les domaines des variables de $\mathcal{V}$.

Une boîte $\mathbf{B} \subseteq \boldsymbol{D}$ est solution du problème MAX-NCSP $\mathcal{S}$ si elle correspond à une affectation des variables de $\mathcal{V}$ dans $\boldsymbol{D}$ telle que :

$$\nexists \mathbf{B}' \subseteq \boldsymbol{D} \text{ t.q. } \mathsf{satis}_C(\boldsymbol{B}') > \mathsf{satis}_C(\boldsymbol{B})$$

où $\mathsf{satis}_C(\boldsymbol{B})$ détermine le nombre de contraintes de $C$ possiblement (ou partiellement) satisfaites par la boîte $\mathbf{B}$. Par possiblement satisfaite, nous entendons qu'il existe dans $\mathbf{B}$ des réels satisfaisant l'ensemble $C$ de contraintes et des réels ne le satisfaisant pas (cf. $j^e$ extension du pavé $\mathbf{P}$ de la figure 3.7). Nous présentons maintenant la définition d'une approximation intérieure d'un problème MAX-NCSP.

**Définition 3.17** (Approximation intérieure d'un problème MAX-NCSP)**.** L'approximation intérieure d'un problème MAX-NCSP $\mathcal{S} = \langle C, \mathcal{V}, \boldsymbol{D} \rangle$ est composée de l'ensemble des boîtes $\mathbf{B} \subseteq \boldsymbol{D}$ telles que :

$$\nexists \mathbf{B}' \subseteq \boldsymbol{D} \text{ t. q. } \mathsf{csatis}_C(\boldsymbol{B}') > \mathsf{csatis}_C(\boldsymbol{B})$$

où $\mathsf{csatis}_C(\boldsymbol{B})$ détermine le nombre de contraintes de $C$ certainement satisfaites par la boîte $\mathbf{B}$. Un pavé est certainement satisfait lorsque tous les réels le composant satisfont l'ensemble $C$ de contraintes.

Calculer une approximation intérieure d'un problème MAX-NCSP consiste à déterminer un ensemble de boîtes maximisant le nombre de contraintes certainement satisfaites du système. Si nous considérons un problème MAX-NCSP $\mathcal{S}$, chacune des boîtes $\mathbf{B}$ de l'espace de recherche initial (*i.e.* $\mathbf{B} \subseteq \boldsymbol{D}$) peut être caractérisée en fonction de trois ensembles disjoints de contraintes : $\mathcal{C} \subseteq C, \mathcal{P} \subseteq C, \mathcal{N} \subseteq C$.

Nous avons alors $\mathcal{C} \cup \mathcal{P} \cup \mathcal{N} = C$ où chaque ensemble correspond respectivement à l'ensemble de contraintes $\mathcal{C}$ertainement satisfaites, $\mathcal{P}$ossiblement satisfaites et $\mathcal{N}$on satisfaites par la boîte $\boldsymbol{B}$. La figure 3.8 illustre la caractérisation $\mathcal{CPN}$ de sous-ensembles d'un espace de recherche $\boldsymbol{D}$ par rapport à trois contraintes $c_1, c_2$ et $c_3$.

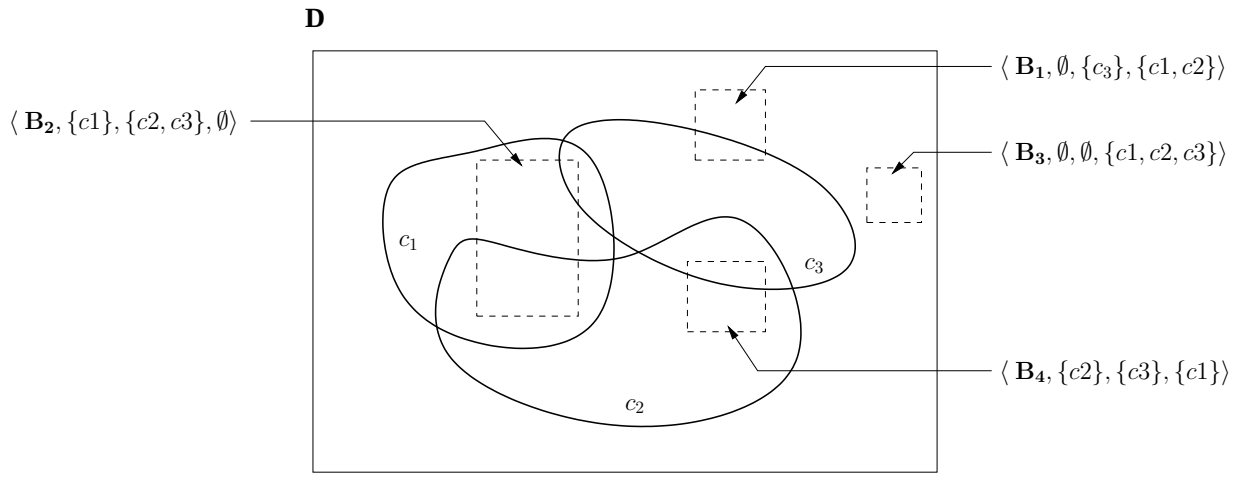



Figure 3.8 – Caractérisation $\mathcal{CPN}$ des pavés $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ et $\mathbf{B}_4$ d'un espace de recherche $\boldsymbol{D}$ en fonction de trois contraintes numériques $c_1, c_2, c_3$.

Afin de réaliser la caractérisation $\mathcal{CPN}$ d'un espace de recherche, nous définissons trois opérateurs $\mathsf{CS}$, $\mathsf{PS}$ et $\mathsf{NS}$ qui calculent respectivement les sous-ensembles de contraintes $\mathcal{C}, \mathcal{P}$ et $\mathcal{N}$ d'une boîte $\mathbf{B}$ de l'espace de recherche d'un problème MAX-NCSP.

**Définition 3.18** (Opérateur $\mathsf{CS}$)**.** Soit $\mathbf{B}$ une boîte et $C$ un ensemble de contraintes numériques. L'opérateur $\mathsf{CS}(C, \boldsymbol{B})$ calcule le sous-ensemble de contraintes $C'$ tel que :

$$C' \subseteq C \text{ et } \forall c \in C', \boldsymbol{B} \subseteq \rho_c$$

**Définition 3.19** (Opérateur $\mathsf{PS}$)**.** Soit $\mathbf{B}$ une boîte et $C$ un ensemble de contraintes numériques. L'opérateur $\mathsf{PS}(C, \boldsymbol{B})$ calcule le sous-ensemble de contraintes $C'$ tel que :

$$C' \subseteq C \text{ et } \forall c \in C', \boldsymbol{B} \not\subseteq \rho_c \wedge \boldsymbol{B} \cap \rho_c \neq \emptyset$$

**Définition 3.20** (Opérateur $\mathsf{NS}$)**.** Soit $\mathbf{B}$ une boîte et $C$ un ensemble de contraintes numériques. L'opérateur $\mathsf{NS}(C, \boldsymbol{B})$ calcule le sous-ensemble de contraintes $C'$ tel que :

$$C' \subseteq C \text{ et } \forall c \in C', \boldsymbol{B} \cap \rho_c = \emptyset$$

Une fois ces trois opérateurs spécifiés, nous proposons la définition d'une boîte « $\mathcal{CPN}$ », qui associe à une boîte les trois ensembles de contraintes $\mathcal{C}, \mathcal{P}$ et $\mathcal{N}$.

**Définition 3.21** (Boîte $\mathcal{CPN}$)**.** Une boîte $\mathcal{CPN}$ est un quadruplet $\langle \boldsymbol{B}, \mathcal{C}, \mathcal{P}, \mathcal{N} \rangle$ défini pour un ensemble de contraintes $C$ et un pavé $\mathbf{B}$ tel que :

$$\begin{aligned} \mathcal{C} &= \mathsf{CS}(C, \mathbf{B}), \\ \mathcal{P} &= \mathsf{PS}(C, \mathbf{B}), \\ \mathcal{N} &= \mathsf{NS}(C, \mathbf{B}) \end{aligned}$$

Nous venons de présenter les concepts de base nécessaires à la modélisation d'un problème MAX-NCSP, concernant la résolution d'un problème de cette classe, nous proposons deux approches. La première consiste en un algorithme naïf d'évaluation-bissection, la deuxième tente d'améliorer les performances obtenues par l'algorithme naïf et consiste en une adaptation d'un algorithme de *branch and bound* couplé aux notions d'extension intérieure et de propagation de contraintes.

#### *3.2.10.2 Résolution d'un* MAX-NCSP

Nous proposons différents algorithmes permettant de calculer l'approximation intérieure d'un problème MAX-NCSP. Le premier consiste en un simple algorithme d'*évaluation-bissection* (*branch and bound*), alors que le deuxième repose sur l'application de l'opérateur d'extension intérieur présenté ci-avant. Enfin, les versions suivantes consistent en des variantes de l'algorithme basé sur l'extension intérieure, nous permettant d'évaluer la pertinence de notre algorithme.

**Algorithme naïf d'évaluation-bissection** Notre première solution destinée à calculer l'approximation intérieure d'un problème MAX-NCSP est basée sur le processus d'*évaluation-bissection*, illustré en figure 3.9, tel qu'il est utilisé dans les travaux de Sam-Haroud et Faltings [SHF94] ou de Jaulin et Walter [JW93] consacrés à l'approximation intérieure d'un NCSP. Le pincipe de ces algorithmes consiste à subdiviser l'espace de recherche jusqu'à un seuil fixé afin de caractériser les boîtes résultant de cette subdivision. La subdivision est stoppée dès lors qu'un pavé satisfait complètement l'ensemble des contraintes (on parle alors de boîte certainement satisfaite) ou lorsque toutes les valeurs de ce pavé violent les contraintes (boîte non satisfaite). Lorsque le seuil minimum de subdivision est atteint, il est possible de rencontrer des pavés pour lesquels certaines valeurs satisfont les contraintes et d'autres non, on parle alors de boîte possiblement satisfaite (cf. figure 3.9).

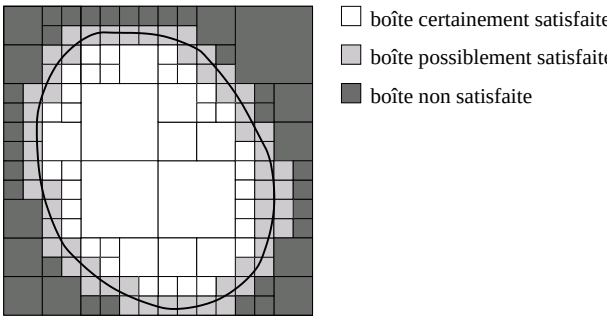



Figure 3.9 – Illustration du processus d'*évaluation-bissection*, d'après Sam-Haroud et Faltings [SHF94].

L'étape d'évaluation consiste à déterminer la consistance de chacune des contraintes par l'application des opérateurs CS, PS et NS à une boîte donnée. L'exploration d'une branche de l'arbre de recherche lors du processus d'évaluation-bissection se termine lorsque la boîte d'intérêt (*i*) ne possède plus de contraintes possiblement satisfaites, *i.e.* toutes les contraintes du système ont été reparties en contraintes certainement satisfaites ou en contraintes non satisfaites, ou (*ii*) est plus petite qu'une taille minimale généralement définie par l'utilisateur. Si lors de l'évaluation d'une boîte le nombre de contraintes partiellement satisfaites est non nul (*i.e.* $|\mathcal{P}| \neq 0$) et que la boîte est de taille supérieure au seuil spécifié, cette dernière est alors subdivisée : c'est l'étape de bissection.

À l'issue de l'algorithme, les zones MAX-NCSP sont constituées en regroupant les nœuds de l'arbre possédant le plus grand nombre de contraintes certainement satisfaites. De plus, l'ensemble de l'espace de recherche est parcouru lors du processus d'évaluation-bissection, la propriété de *complétude* étant ainsi assurée.

L'implémentation de cette approche est décrite dans l'algorithme AlBissectionMaxCSP présenté en table 3.5. Ce dernier assure aux solutions la propriété de *correction*. Il est paramétré par un ensemble $C$ de contraintes et par une boîte $\mathbf{B}$ représentant l'espace de recherche initial. Il calcule ensuite la liste des pavés composant les meilleures approximations intérieures du problème MAX-NCSP. En effet, la méthode meilleuresBoîtes retourne les meilleures boîtes en termes de satisfaction certaine des contraintes du problème (*i.e.* les boîtes qui maximisent le nombre de contraintes certainement satisfaites : $|\mathcal{C}|$).

Les algorithmes 3.5 et 3.6 utilisent une méthode de découpe des boîtes, ainsi qu'un opérateur permettant de réaliser la différence entre deux pavés. Ces deux opérations sont décrites à la suite de la présentation des algorithmes.

---

ALG. 3.5 – Algorithme naïf de calcul d'approximation intérieure pour un problème MAX-NCSP par évaluation-bissection.

```
1  AlBissectionMaxCSP(in: C, B ∈ 𝕀^n;
                      out: ℛesultℒ : liste < boîtes 𝒞𝒫𝒩 >)
2  begin
3   ℛesultℒ ← ∅
4   ℒ ← {⟨B, CS(C), PS(C), NS(C)⟩}
5   while (non estVide(ℒ))
6     ⟨B, cs, ps, ns⟩ ← récupèreBoîte(ℒ)
7     if (|ps| = 0) then
8       ℛesultℒ ← ℛesultℒ ∪ ⟨B, cs, ps, ns⟩
9     else
        % il reste des contraintes partiellement satisfaites dans B
10      ℒ ← ℒ ∪ bissection(⟨B, cs, ps, ns⟩)
11    end
12  endwhile
13  return meilleuresBoîtesℛesultℒ
14 end
```

---

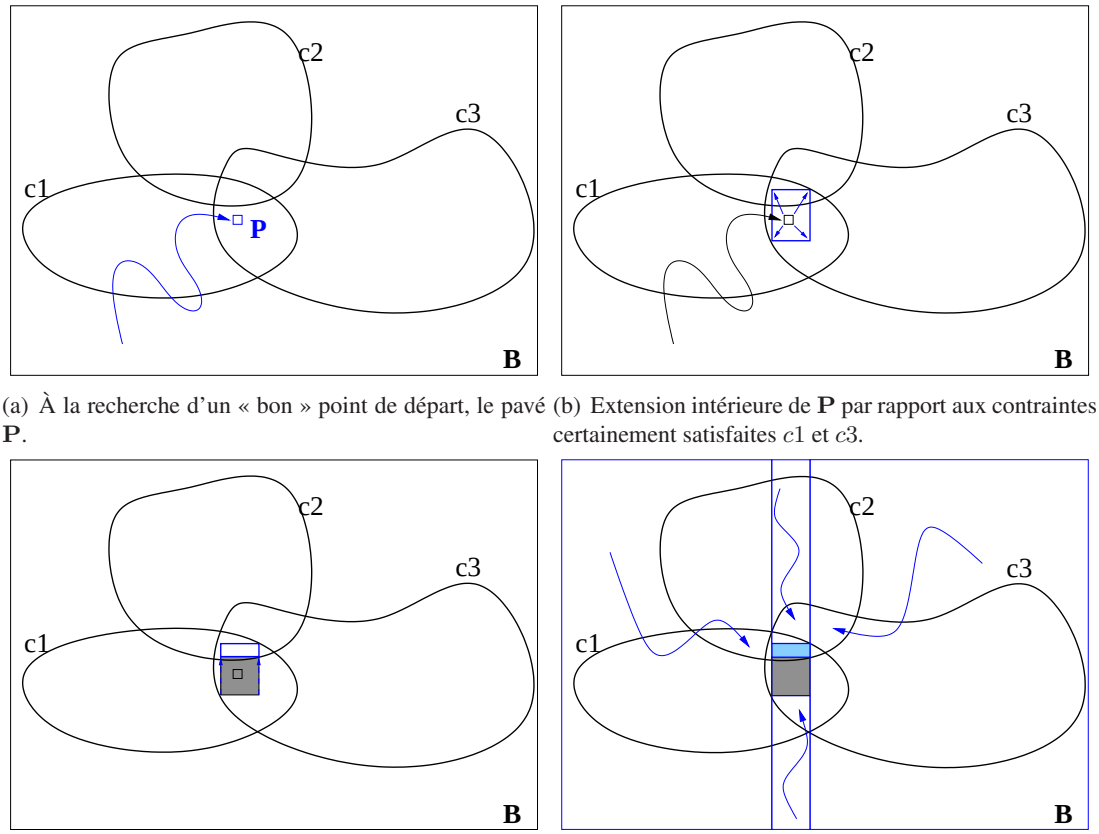
**Approche par extension intérieure** Afin d'améliorer les performances de résolution de l'algorithme AlBissectionMaxCSP, nous proposons d'avoir recours à l'opérateur d'extension intérieure présenté en section 3.2.8. Cette extension est utilisée dans une approche de type évaluation-bissection qui maintient à jour une valeur $m$ représentant un optimum du nombre de contraintes certainement satisfaites du système de contraintes $C$. À l'instar d'un processus de résolution classique, nous cherchons à élaguer le plus tôt possible les branches de l'arbre de recherche qui ne peuvent améliorer la meilleure solution courante. Ces branches sont représentées par les boîtes $\mathbf{B}$ telles que $|\mathsf{PS}(C, \boldsymbol{B})| + |\mathsf{CS}(C, \boldsymbol{B})| < m$. Dans ce cas, nous sommes assurés que les pavés contenus dans ces branches ne peuvent être meilleurs que l'optimum

courant, nous pouvons ainsi les retirer du processus de résolution.

Le principe de cet algorithme, qui cherche à calculer l'approximation intérieure d'un problème MAX-NCSP composé d'un ensemble $C$ de contraintes dans un espace initial de recherche $\mathbf{B}$, est le suivant :

1. déterminer un pavé canonique $\mathbf{P} \subseteq \mathbf{B}$ (*i.e.* intérieur à une des contraintes de $C$),
2. calculer une extension intérieure $\mathbf{P}'$ de $\mathbf{P}$ basée sur les contraintes certainement satisfaites afin de maximiser la zone prometteuse,
3. en prenant uniquement en compte les contraintes possiblement satisfaites, effectuer une approximation extérieure afin de se concentrer sur les configurations proches de $\mathbf{P}'$. Les régions de l'espace restantes sont traitées ultérieurement afin d'assurer une exploration complète de l'espace de recherche.

La figure 3.10 illustre les différentes étapes du processus de résolution pour un ensemble de trois contraintes $c1, c2$ et $c3$.




(a) À la recherche d'un « bon » point de départ, le pavé $\mathbf{P}$.

(b) Extension intérieure de $\mathbf{P}$ par rapport aux contraintes certainement satisfaites $c1$ et $c3$.

(c) Réduction autour de la région prometteuse par propagation des contraintes potentiellement satisfaites : $c2$.

(d) Itération sur les domaines restants, rajout de la zone propagée à la liste des boîtes à traiter et abandon de la boîte grisée.

Figure 3.10 – Illustration du processus de résolution d'un problème MAX-NCSP.

ALG. 3.6 – Algorithme AIMaxNCSP d'approximation intérieure pour un problème MAX-NCSP.

```
1  AIMaxNCSP(in: C, B ∈ 𝕀^n;
                 out: ℛesultℒ: list < boîtes𝒞𝒫𝒩)
2  begin
3   maxSat ← 0
4   ℒ ← {⟨B, CS(C), PS(C), NS(C)⟩}
5   while (non estVide(ℒ))
6     ⟨B, c, p, n⟩ ← récupèreBoîte(ℒ)
      % recherche d'un pavé canonique dans la boîte B
9     ⟨P, cs, ps, ns⟩ ← récupèreConfiguration(B)
11    if (|cs| > maxSat) then
        % mise à jour de l'optimum maxSat du nombre de contraintes certainement satisfaites
12      maxSat ← |cs|
13    end
14    if (|cs + ps| ≤ maxSat) then
18      if (|cs| > 0) then
          % Extension intérieure de l'ensemble des contraintes certainement satisfaites
19        B' ← AlgoAEI(cs, P, B, ε)
19        cs' ← CS(C, B')
19        ps' ← PS(C, B')
19        ns' ← NS(C, B')
15        if (|ps'| = 0) then
16          ℛesultℒ ← ℛesultℒ ∪ ⟨B', cs', ps', ns'⟩
17        else
            % Propagation des contraintes potentiellement satisfaites
20          B'' ← propagation(B', ps')
21          if (B'' = ∅) then
22            if (|cs'| + |ps'| > maxSat) then
23              ℒ ← ℒ ∪ bissection(B')
24            end
25          else
26            ℒ ← ℒ ∪ ⟨B'', CS(C, B''), PS(C, B''), NS(C, B'')⟩
27            ℒ ← ℒ ∪ {B'\B''}
28            ℒ ← ℒ ∪ {B\B'}
29          end
25        end
30      else
31        ℒ ← ℒ ∪ bissection(B)
32      end
            % |cs + ps| < maxSat
            % la boîte ne peut être meilleure que l'optimum courant : nous ne la traitons pas
34    end
35  endwhile
36  return meilleuresBoîtes(ℛesultℒ)
37 end
```

L'approche par extension intérieure est présentée dans l'algorithme 3.6. Considérons une boîte $\mathcal{CPN}$ $\langle \boldsymbol{B}, cs, ps, ns \rangle$ (cf. ligne 6) obtenue à une étape de l'application de l'algorithme. En premier lieu, nous obtenons un point $\mathbf{P}$ (*i.e.* une boîte canonique) par application d'une procédure récupèreConfiguration qui retourne un pavé canonique à l'intérieur du pavé courant. Dans le cas où ce dernier est meilleur (c'est-à-dire qu'il satisfait plus de contraintes que l'optimum courant), nous mettons à jour la valeur de $m$. Nous appliquons ensuite l'opérateur d'extension intérieure dans le but de maximiser $\mathbf{P}$, puis nous pratiquons une étape de propagation des contraintes possiblement satisfaites sur la boîte étendue.

La version de l'algorithme que nous proposons intègre un certain nombre d'améliorations par rapport à la version classique du processus de *branch and bound* :

- l'évaluation d'un pavé canonique $\mathbf{P}$ choisi dans l'espace de recherche initial $\mathbf{B}$ afin de mettre à jour l'optimum $m$ le plus souvent possible. En effet, une boîte canonique va favoriser très fortement le fait de ne pas contenir de contraintes possiblement satisfaites (de par la taille très réduite des domaines) et donc permettre d'élaguer les branches inintéressantes plus tôt dans le processus de recherche.
- le calcul d'une boîte $\mathbf{B'}$ par application de l'opérateur d'extension intérieure au pavé canonique $\mathbf{P}$ afin de réduire la taille de l'espace de recherche.
- l'application de l'opérateur d'approximation extérieure autour de la boîte $\mathbf{B'}$ par rapport à l'ensemble $\mathcal{P}$ des contraintes possiblement satisfaites. Se concentrer cet ensemble de contraintes offre en effet les avantages suivants :
  - si la propagation échoue (inconsistance par rapport aux contraintes possiblement satisfaites), alors la boîte ne peut être meilleure (en termes de contraintes certainement satisfaites) que : $|\mathsf{CS}(C, \boldsymbol{B'})| + |\mathsf{PS}(C, \boldsymbol{B})|$.
  - si la propagation réduit partiellement les domaines, la pavé ainsi obtenu est alors une zone prometteuse de l'espace de recherche.

Il apparaît évident que les performances de cet algorithme dépendent très fortement de l'étape à laquelle la boîte optimale (*i.e.* qui possède le plus grand nombre de contraintes certainement satisfaites) est déterminée. En effet, plus cette dernière est trouvée tôt, plus les branches de l'arbre de recherche sont élaguées rapidement. Il est donc primordial de déterminer une méthode récupèreConfiguration efficace afin d'obtenir le pavé canonique optimal le plus tôt possible et par là même d'optimiser les performances de l'algorithme. C'est dans cette optique que nous avons développé un algorithme de recherche locale (RL) continue permettant de déterminer très rapidement un « bon » point de départ de l'algorithme, *i.e.* un pavé satisfaisant certainement (au sens de l'opérateur CS) un maximum de contraintes du problème. La présentation de cet algorithme de recherche locale continue a lieu en section 3.2.13.4.

Afin d'illustrer notre approche, nous proposons différentes versions de l'algorithme 3.6, à savoir :

- une version basique BoîteBissectEvalMaxNCSP qui ne détermine pas la pertinence de la boîte courante par un pavé canonique, mais en évaluant le triplet $\langle cs, ps, ns|$ directement sur le pavé courant. De plus, l'algorithme BoîteBissectEvalMaxNCSP n'effectue pas d'extension intérieure, ni de propagation de contraintes.
- une version avec évaluation de la boîte courante en étudiant le pavé canonique central et sans extension intérieure ni propagation. Cet algorithme PointCentralBissectEvalMaxNCSP génère ainsi le pavé canonique correspondant aux valeurs médianes des domaines de définition de chacun des variables impliquées dans les contraintes du problème.
- une version RLBissectEvalMaxNCSP qui génère le point d'intérêt par une méthode de recherche locale continue, afin d'atteindre des zones prometteuse plus rapidement. Comme les deux algorithmes précédents, cette version n'effectue ni extension intérieure, ni propagation de contraintes.
- une version AIRLBissectEvalMaxNCSP, qui génère le pavé canonique à l'aide d'une méthode

de recherche locale continue, mais qui effectue ensuite un pas d'extension intérieure concernant les contraintes certainement satisfaites, afin de maximiser la zone prometteuse avant de poursuivre la résolution.

- enfin, la version AIPropagRLBissectEvalMaxNCSP combine génération du pavé canonique par méthode de recherche locale, extension intérieure basée sur les contraintes certainement satisfaites et propagation de contraintes possiblement satisfaites. Cet algorithme correspond à l'agorithme présenté en table 3.6 et combine toutes les améliorations proposées.

Toutefois, avant de nous consacrer à la présentation d'une extension continue de la recherche locale, nous présentons la méthode split, ainsi que l'opérateur $\setminus$ défini sur des boîtes utilisés dans les algorithmes 3.5 et 3.6.

**Méthode de découpe (bissection) d'une boîte $\mathcal{CPN}$** L'opération de découpe d'une boîte $\mathcal{CPN}$ telle qu'utilisée dans les algorithmes précédents correspond à une étape classique de bissection des méthodes de résolution de contraintes, la seule spécificité est de tenir compte des ensembles de contraintes satisfaites, non satisfaites et partiellement satisfaites. La bissection s'effectue selon une heuristique de découpe (nous avons choisi de découper toujours selon la variable de domaine maximum). Une fois la boîte $\mathcal{CPN}$ $\mathbf{B}$ courante découpée en deux sous-boîtes $\mathbf{B}_1$ et $\mathbf{B}_2$, nous devons générer les ensembles $\mathcal{C}, \mathcal{P}$ et $\mathcal{N}$ associés. Ces derniers sont obtenus par application des opérateurs CS, PS et NS aux deux boîtes $\mathbf{B}_1$ et $\mathbf{B}_2$.

Nous obtenons donc deux nouvelles boîtes $\mathcal{CPN}$ définies par les quadruplets : $\langle \mathbf{B}_1, cs_1, ps_1, ns_1 \rangle$ et $\langle \mathbf{B}_2, cs_2, ps_2, ns_2 \rangle$. Où $cs_1, ps_1$ et $ns_1$ (resp. $cs_2, ps_2$ et $ns_2$) sont obtenus par application des opérateurs CS, PS et NS à la boîte $\mathbf{B}_1$ (resp. $\mathbf{B}_2$). Celles-ci sont ensuite rajoutées à la liste des boîtes non traitées afin d'assurer une exploration complète de l'espace de recherche initial.

**Opérateur de différence entre deux boîtes** L'opération réalisant la différence entre deux boîtes $\mathbf{B}$ et $\mathbf{B'}$ ($\mathbf{B'} \subseteq \mathbf{B}$) consiste à générer l'ensemble des boîtes résultant de la soustraction de $\mathbf{B'}$ à $\mathbf{B}$. Une illustration de ce processus en deux dimensions (*i.e.* pour deux variables) est donnée en figure 3.11.

La différence $\mathbf{B} \setminus \mathbf{B'}$ consiste donc à découper les domaines des variables de manière à ce que leurs intersections soient vides. Cette opération conduit donc à la création de nouvelles boîtes aux domaines réduits par rapport à la boîte englobante ($\mathbf{B}$). L'opération $\mathbf{B} \setminus \mathbf{B'}$, illustrée en figure 3.11, conduit à la création de quatre nouvelles boîtes $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$ et $\mathbf{B}_4$ telles que l'intersection des domaines des variables $v_1$ et $v_2$ soit vide.

### 3.2.11 Une extension continue du cadre de la recherche locale

Notre approche MAX-NCSP (cf. algorithme 3.6) nécessite, pour être performant, d'obtenir le plus rapidement possible un « bon » point de départ en termes de satisfaction des contraintes du problème. Nous proposons donc d'instancier la méthode récupèreConfiguration de l'algorithme 3.6 par une procédure de recherche locale continue.

Avant de décrire en détails le cadre continu que nous avons proposé pour adapter les procédures classiques de recherche locale discrète, nous consacrons une sous-section à un état de l'art concis des méthodes incomplètes de résolution de contraintes dans les domaines continus, dont fait partie la recherche locale. Nous détaillons ensuite l'adaptation continue que nous proposons pour chacune des entités clés d'un algorithme de recherche locale, puis nous présentons l'algorithme de recherche taboue continue utilisé dans la méthode de résolution MAX-NCSP des problèmes de placement de caméra en environnements virtuels.

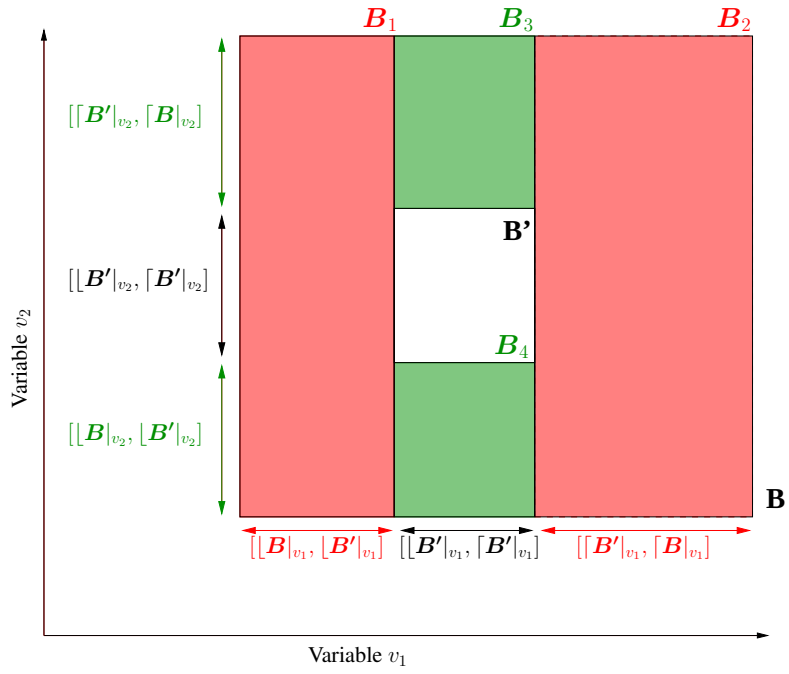



Figure 3.11 – Différence entre deux boîtes $\mathbf{B}$ et $\mathbf{B'}$ en deux dimensions.

### *3.2.11.1 État de l'art des méthodes de recherches incomplètes*

Les méthodes de résolution incomplètes ont été appliquées avec succès à un grand nombre de CSP et de problèmes d'optimisation à domaines discrets, citons par exemple les articles de Hao *et al.* [HGH98] ou plus récemment de Blum et Roli [BR03]. Parmi elles, les algorithmes de recherche locale sont basés sur un processus itératif d'amélioration d'une configuration initiale choisie aléatoirement. L'amélioration d'une configuration est relative à une notion de fonction de pénalité $f$ (aussi appelée fonction de coût ou fonction d'erreur), qui représente le plus souvent dans le cadre des CSPs, le nombre de contraintes violées par la configuration courante (selon l'idée du *Min-Conflict* [MJPL92]). La recherche locale est de plus en plus considérée comme une solution de choix pour la résolution de problèmes de satisfaction de contraintes, et ce en particulier grâce au développement du système COMET [MvH02].

Le principe des algorithmes de recherche locale est de choisir une configuration initiale aléatoire, d'explorer un voisinage de cette configuration et de modifier cette dernière sous certaines conditions (*e.g.* combinaison d'amélioration de la pénalité et d'une métaheuristique). Ces étapes sont répétées jusqu'à satisfaction d'une condition de terminaison de l'algorithme (une solution a été trouvée ou un nombre maximum d'itérations a été atteint).

La recherche locale (ou plus généralement les métaheuristiques) est guidée par deux notions complémentaires : l'*intensification* et la *diversification*. La dernière consiste à garantir que la recherche explore le plus largement possible l'espace de recherche, sans délaisser des zones au profit d'autres. Ceci est généralement assuré par une réinitialisation aléatoire de la configuration de départ de l'algorithme. *A contrario*, l'intensification consiste à approfondir la recherche de solutions dans une zone supposée prometteuse.

Afin d'éviter un « bouclage » de l'algorithme de recherche locale, *i.e.* de retomber sur des configurations déjà explorées, une partie métaheuristique peut être ajoutée. Elle permet alors d'assurer certaines propriétés à l'algorithme par rapport à l'espace de recherche (et donc au niveau de l'intensification et de la diversification). L'une des métaheuristiques les plus efficaces est certainement la recherche taboue proposée par Glover [Glo86, GL97] et séparément par Hansen [Han86]. Elle consiste en la création d'une liste à court terme stockant les configurations récemment étudiées. Celles-ci sont alors interdites pendant un nombre fixe d'itérations correspondant à la taille de la liste taboue (*tabu tenure*). L'algorithme évite alors de boucler en étudiant d'autres régions de l'espace de recherche amenant à la création d'autres voisinages.

Bien qu'étant très répandues dans les domaines discrets, l'application des métaheuristiques aux CSPs à domaines continus reste très marginale. Toutefois, des travaux ont été proposés concernant l'adaptation de méthodes populationnistes aux domaines continus, en particulier les travaux de Michalewicz [Mic95]. Chelouah et Siarry ont étendus de nombreuses métaheuristiques au cas continu, en particulier ils ont proposé une version continue de la recherche taboue : *ECTS – Enhanced Continuous Tabu Search* – [CS00]. Une des difficultés liées à la transposition des méthodes discrètes aux domaines continus réside dans la prise en compte de la taille des domaines des variables, ceux-ci étant bien plus vastes dans le cas continu que dans le cas discret. Pour résoudre ce problème, *ECTS* utilise une métrique définissant des rectangles $n$-dimensionnels concentriques autour de la configuration courante, dans lesquels les configurations voisines sont déterminées aléatoirement. Les mêmes auteurs ont également proposé la méthode *CHA* [CS03] (*Continuous Hybrid Algorithm*) qui hybride un algorithme génétique avec une méthode de recherche locale de type du simplexe de Nelder-Mead.

Une autre adaptation continue de la recherche taboue a été proposée par Battiti et Tecchiolli [BT96] par le biais d'un algorithme hybride. Dans cette approche, les domaines continus sont traités par discrétisation comme une grille de points recouvrant l'espace de recherche $\mathbb{R}^n$. Anglada *et al.* [ACZ04] proposent également un algorithme continu proche de la recherche taboue : la recherche adaptative. Dans leur solution, le choix de la variable à modifier dépend de la satisfaction des contraintes. Chacune des contraintes est associée à une fonction de coût représentant le degré de satisfaction de la contrainte par la configuration courante. Une fois toutes les contraintes évaluées par leurs fonctions d'erreur respectives, les auteurs combinent pour chaque variable toutes les fonctions d'erreur des contraintes dans lesquelles cette variable apparaît. Enfin, la variable d'erreur maximale est désignée comme variable « coupable » et sa valeur est modifiée. La nouvelle valeur est celle pour laquelle l'erreur totale de la prochaine affectation complète des variables est minimale (à la manière d'une descente sous gradients). Afin de ne pas tomber dans des minima locaux, la recherche adaptative est munie d'une mémoire stockant les variables interdites, comme pour une recherche taboue.

L'heuristique GRASP a également été très largement utilisée pour résoudre des problèmes d'optimisation discrets. GRASP (*Greedy Randomized Adaptative Search Procedures*) a été proposée par Feo et Resende [FR89, FR95] et consiste en un processus d'applications successives (ou de redémarrages successifs à partir d'un nouveau point de départ) d'une itération elle même décomposée en deux phases. La première est une phase de construction lors de laquelle une solution réalisable est construite. La deuxième correspond en une étape de recherche locale dans laquelle un optimum local appartenant au voisinage de la solution courante est recherché. La métaheuristique consiste donc en l'application successives de ces deux phases à partir de points de départ différents. La meilleure solution globale est finalement retournée comme résultat. Hirsch *et al.* [HMPR07] ont proposé une adaptation continue de GRASP, *Continuous*-GRASP , ou plus simplement C-GRASP pour le domaine de l'optimisation continue. L'ensemble des configurations continues dans C-GRASP est de la forme $\mathcal{S} = \{x = (x_1, \ldots, x_n) \in \mathbb{R}^n : l \leq x \leq u\}$ avec $l, u \in \mathbb{R}^n$ et tels que $l \leq u$. Le problème consiste ensuite à trouver la configuration optimale dans

$\mathcal{S}$ par rapport à une fonction de coût $f$. Le principe de C-GRASP est identique à celui de GRASP, C-GRASP est une métaheuristique appliquée à une recherche stochastique multiple utilisant une procédure aléatoire gloutonne pour générer les points de départ de la recherche. La différence principale réside dans la composition d'une itération de C-GRASP. En effet, une itération de GRASP consiste comme nous l'avons dit en une construction unique gloutonne aléatoire d'un point de départ suivie d'une procédure d'amélioration locale. Au contraire, dans C-GRASP, une itération consiste plutôt en une série de cycles de construction de solution et d'améliorations locales imbriquées. Comme dans GRASP, la solution générée lors de la construction est ensuite améliorée localement, puis sert de point de départ à la construction d'une nouvelle solution. L'amélioration locale est donc réutilisée directement dans la boucle afin d'obtenir une nouvelle solution. Dans GRASP un point de départ unique donne lieu à une série d'améliorations locales, alors que dans C-GRASP, un point de départ donne lieu à une amélioration qui entraîne à son tour la construction d'une nouvelle solution. L'algorithme que nous présentons (cf. section 3.2.13.4) est assez proche de l'idée de la métaheuristique GRASP, il contient également une double boucle de génération-amélioration des méthodes, mais contrairement à C-GRASP, notre approche est basée sur une représentation continue des réels (*i.e.* les intervalles).

Dans les travaux sus-cités, les configurations étudiées sont, comme dans le cas discret, des affectations complètes (continues) de variables, *i.e.* des points de $\mathbb{R}^n$. Les notions de voisinage et de métaheuristiques des algorithmes discrets doivent donc être réécrites pour tenir compte des spécificités des domaines continus (leur très grande taille, *i.e.* $\mathbb{R}$ discrétisé). Nous proposons une approche orthogonale. En effet, pour les besoins de notre application (le placement de caméra en environnements virtuels), nous devons considérer les configurations non pas comme des points de $\mathbb{R}^n$, mais comme des intervalles de $\mathbb{I}^n$. Nous avons donc redéfini les notions de configurations, de voisinage et d'évaluation des algorithmes classiques afin de prendre en compte la nature continue des contraintes et des configurations utilisées. Notre méthode peut être rapprochée des travaux de Barichard et Hao [BH03] qui proposent PICPA, une méthode hybride de résolution de problèmes sous contraintes, dont la partie métaheuristique est gérée par un algorithme génétique. Dans PICPA, les individus sont des boîtes sélectionnées selon un critère multi-objectif, dans notre méthode les configurations sont des pavés sélectionnés selon différents critères.

### 3.2.12 La recherche taboue

Les méthodes de recherche locale appliquées à la résolution de CSPs discrets (cf. définition 3.1) sont généralement définies par trois composantes :

- une fonction de coût (appelée également fonction de pénalité ou d'erreur) $f : D_1 \times \cdots \times D_n \rightarrow \mathbb{N}$ à maximiser ou minimiser selon les approches,
- une fonction de voisinage qui, étant donné une configuration courante, fournit un ensemble de « voisins », c'est-à-dire un ensemble de configurations atteignables depuis la configuration courante,
- une métaheuristique destinée à guider la recherche, en particulier à lui éviter de rester bloquer dans les optima locaux et donc à éviter le « bouclage » de l'algorithme.

Il est à noter que les notions de configurations (c'est-à-dire une affectation partielle ou complète des variables du problème) et d'évaluation de ces configurations par les fonctions de pénalité sont évidentes lorsque nous nous plaçons dans le cas discret. De même, les notions d'appartenance d'une configuration à la liste ou de choix aléatoire ne sont que rarement définies dans les travaux n'étant pas ambigus pour des CSP discrets. Toutefois, une fois transposées au domaines continus, ces notions méritent d'être

formalisées à nouveau.

Dans cette thèse, nous nous intéressons uniquement à la recherche taboue comme métaheuristique destinée à guider la recherche et nous considérons taboues les configurations visitées (à l'inverse de certaines approches qui marquent les mouvements comme tabous). Les configurations sont donc marquées comme taboues une fois qu'elles ont été visitées et par conséquent stockées dans la liste taboue $\mathcal{T}$ pour un nombre $\mathtt{t}$ (la taille de la liste taboue ou *tabu tenure*) d'itérations de l'algorithme. Une fonction de mise à jour de la liste taboue est utilisée à chaque nouvelle itération pour actualiser les nombres d'itérations restants pour chaque configuration et pour retirer le dernier élément de $\mathcal{T}$. À tout moment, la liste taboue représente donc un instantané des $\mathtt{t}$ derniers mouvements effectués par l'algorithme dans l'espace de recherche. Ce comportement basique d'une recherche taboue peut être amélioré par ajout de mécanismes de diversification et/ou d'intensification.

Comme nous venons de le présenter, les composantes d'une recherche locale taboue appliquée aux CSP discrets sont clairement identifiées et définies. De plus, les travaux de recherche ayant tenté d'adapter la recherche taboue (RT) aux domaines continus (en particulier ceux de Chelouah et Siarry [CS00, CS03], voir ci-avant) se sont contentés de discrétiser l'espace de recherche et perdent donc la notion intrinsèque de continuité de ces problèmes. Afin de prendre en compte les spécificités des CSP continus, nous proposons de définir une recherche locale taboue basée sur les notions d'intervalles à bornes flottantes permettant de capturer les notions de continuité des domaines de définition des variables.

### 3.2.13 Une extension de la recherche locale aux domaines continus

Il apparaît évident que certaines notions triviales pour le cas discret (configuration, voisinage, etc.) ne le sont pas pour les domaines continus. Nous identifions ici les difficultés liées à l'adaptation de ces composantes aux CSP continus et les formalisons.

L'objectif est de permettre d'adapter les différentes notions nécessaires à une recherche taboue pour des représentations diverses des domaines continus, par exemple des discrétisations de l'espace de recherche (*e.g.* $\mathbb{N}^n$ ou $\mathbb{F}^n$) ou pour des intervalles à bornes flottantes de $\mathbb{I}^n$.

Nous définissons dans les sections suivantes les notions nécessaires à l'implémentation d'une méthode de recherche taboue continue, à savoir : les notions de configuration, de fonction de voisinage et de fonction de pénalité. Nous proposons ensuite une instanciation de ce cadre formel aux domaines continus ainsi qu'une implémentation.

#### *3.2.13.1 Une configuration continue*

Une configuration correspond à une affectation complète des variables. Pour les domaines continus, une représentation de l'espace de configurations atteignables doit donc être choisie. Dans la suite de cette section, nous distinguons trois cas de figure possibles :

- une représentation en nombres flottants ($\in \mathbb{F}^n$),
- une représentation par pavés canoniques (cf. définition 3.6),
- une représentation par intervalles à bornes flottantes (cf. définition 3.5).

Afin d'assurer la diversification (une exploration équitable de l'espace de recherche), un opérateur de génération aléatoire de configurations avec une distribution de probabilités sans biais doit être défini.

**Définition 3.22** (Configuration continue)**.** Une configuration continue d'un CSP $\langle \mathcal{C}, \mathcal{V}, \mathbf{D} \rangle$, est définie comme suit :

$$\langle d_1 \times \cdots \times d_n \rangle | \forall i \in [1..n], d_i \subseteq D_i$$

où $D_i$ représente l'ensemble des nombres flottants $\mathbb{F}^n$ dans les cas discrétisant $\mathbb{R}$ ou l'ensemble des intervalles à bornes flottantes $\mathbb{I}^n$ dans les cas continus.

#### *3.2.13.2 Un voisinage continu*

Le concept de voisinage correspond dans le cas discret à un ensemble de taille raisonnable de configurations « relativement proches » de la configuration courante. La notion de proximité dépend bien évidemment de la métrique utilisée. Celle-ci est souvent considérée comme la distance de Hamming, la proximité consistant alors à se situer à une distance de 1 de la configuration courante. Le « voisin » diffère donc de la configuration courante uniquement par la valeur d'une des variables du problème. La taille du voisinage peut être un facteur critique de la recherche. En effet, un voisinage trop réduit limite les améliorations possibles, tandis qu'un voisinage trop important est coûteux à explorer.

Il est possible de considérer un très grand nombre de voisinages pour une configuration, nous proposons de nous limiter à deux sortes de voisinages :

- le voisinage de variable : consiste à modifier la valeur d'une seule variable de la configuration courante, les valeurs des autres variables restant identiques. La sélection de la variable à modifier peut être effectuée selon plusieurs heuristiques : *round robin*, variable de plus grand domaine, choix aléatoire, etc.
- le voisinage de domaines : consiste à autoriser la modification (éventuelle) de toutes les variables de la configuration courante. Les nouvelles valeurs des variables sont alors choisies dans un sous-ensemble de leur domaine de définition afin de respecter une proximité.

La génération du voisinage de la configuration courante est effectuée par le biais d'une fonction $\mathsf{voisinage} : \mathbf{D} \rightarrow 2^{\mathbf{D}}$. Cette fonction, instanciée à la notion de voisinage de variable, est définie comme suit :

$$\mathsf{voisinage}_V(\langle v_1, \ldots, v_i, \ldots, v_n\rangle) = \{\langle v_1, \ldots, v'_i, \ldots, v_n\rangle | \forall i \in [1..n], v'_i \in D_i, v'_i \neq v_i\}$$

Le voisinage de domaines est quant à lui défini de la manière suivante :

$$\mathsf{voisinage}_D(\langle D_1, \ldots, D_i, \ldots, D_n\rangle) = \{\langle D'_1, \ldots, D'_i, \ldots, D'_n\rangle | \forall i \in [1..n], D_i \subseteq D'_i\}$$

Le problème de l'adaptation du concept de voisinage aux domaines continus est que nous risquons de générer des espaces voisins beaucoup trop étendus (*e.g.* exploration de $\mathbb{F}$) pour lesquels l'exploration est trop coûteuse. Nous proposons donc d'ajouter à la fonction voisinage une deuxième fonction permettant de choisir à l'intérieur de ce volume un nombre de voisins représentatifs : $\mathsf{échantillonVoisinage} : \mathbb{Z} \times \mathsf{Im}(\mathsf{voisinage}) \rightarrow 2^{\mathbf{D}}$.

La génération du voisinage d'une configuration comporte donc deux phases :

1. une phase de subdivision de l'espace de recherche en une région contenant les configurations éligibles au statut de voisin (fonction voisinage),
2. la génération, à l'intérieur de cette zone de voisins potentiels, d'un ensemble de $nb_{vois}$ configurations considérées comme *voisins* de la configuration courante.

Enfin, afin d'accroître le mécanisme d'intensification de l'algorithme, nous proposons de définir un facteur d'échelle $\alpha$ permettant de réduire le voisinage à chaque itération de recherche locale. Ainsi, nous nous concentrons progressivement sur un voisinage de plus en plus réduit autour de la configuration courante. Cette notion d'$\alpha$-réduction du voisinage favorise une exploration en profondeur d'une sous-région de l'espace de recherche prometteuse en termes de solutions au problème. La figure 3.12 illustre cinq itérations successives d'une recherche avec voisinage $\alpha$-réduit.

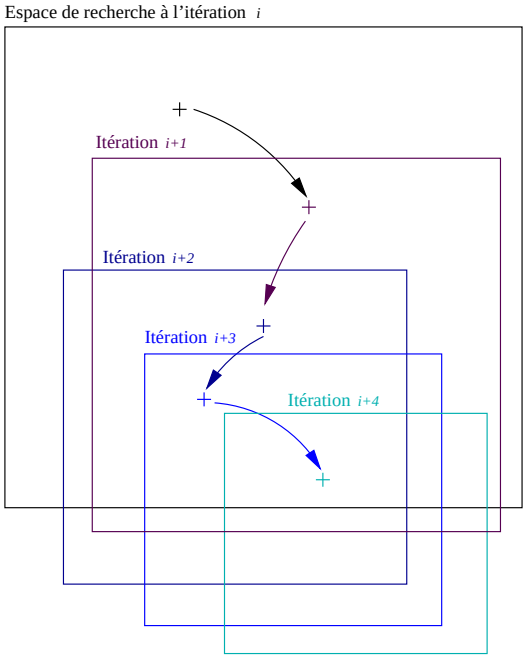



Figure 3.12 – $\alpha$-réduction du voisinage d'une configuration au cours d'une recherche locale.

Nous pouvons rapprocher notre définition de voisinage par $\alpha$-réduction de la notion de *Variable Neighbourhood Search* (VNS) introduite par Mladenović et Hansen [MH97]. L'$\alpha$-réduction offre une certaine flexibilité dans la paramétrisation de la recherche, en effet, si nous souhaitons conserver un voisinage de taille constante, il suffit alors de spécifier $\alpha = 1$. Toutefois, un des inconvénients liés à l'utilisation des techniques de recherche locale consiste en la difficulté du réglage des paramètres de ces méthodes. Nous offrons donc à l'utilisateur une certaine souplesse dans la définition du voisinage d'une configuration qui est contrebalancée par la nécessité de régler le facteur $\alpha$.

#### *3.2.13.3 Une fonction de pénalité continue*

La fonction de pénalité coût permet d'évaluer la qualité d'une configuration en termes de satisfaction de contraintes, *i.e.* elle permet de déterminer le niveau de violation ou de satisfaction des contraintes par une configuration. La définition d'une fonction de pénalité doit respecter deux prérequis :

- l'évaluation d'une configuration par la fonction de pénalité doit être réalisable efficacement (si possible de façon incrémentale en s'appuyant sur la notion de voisinage),
- les évaluations obtenues par application de la fonction de pénalité doivent être comparables.

Ces deux conditions sont aisément satisfaisables lorsque nous travaillons sur $\mathbb{N}$ ou $\mathbb{R}$, elles le deviennent nettement moins lorsque nous nous intéressons à des domaines continus, en particulier lorsque ceux-ci sont représentés par des intervalles à bornes flottantes. En effet, l'évaluation de la satisfaction d'une boîte en termes de contraintes du système peut amener à trois cas distincts :

- la boîte satisfait totalement une contrainte,
- la boîte satisfait seulement partiellement la contrainte,

- la boîte ne satisfait pas la contrainte.

Ces trois cas de figure correspondent aux définitions énoncées dans la section 3.2.10.1 concernant les opérateurs $\mathsf{CS}, \mathsf{PS}$ et $\mathsf{NS}$ des problèmes MAX-NCSP. Nous obtenons donc pour chaque configuration un triplet $(cs, ps, ns)$ représentant le nombre de contraintes certainement satisfaites, partiellement satisfaites et non satisfaites du système de contraintes. Remarquons que nous avons également $cs + ps + ns = n$, avec $|\mathcal{C}| = n$.

Une fois ce triplet de valeurs obtenu pour chacune des configurations évaluées par la fonction de pénalité, nous devons être en mesure de les comparer afin de déterminer le voisin le plus prometteur. Nous devons définir un ordre total $\prec_{\mathbb{I}}$ sur ces triplets, celui de valeur minimale étant le plus intéressant. Nous identifions trois cas :

1. privilégier les configurations satisfaisant certainement le plus de contraintes possibles. Nous définissons alors $\prec_{\mathbb{I}}$ comme : $(cs_1, ps_1, ns_1) \prec_{\mathbb{I}} (cs_2, ps_2, ns_2) \equiv cs_1 > cs_2$.
2. privilégier la capacité d'une configuration à satisfaire de nouvelles contraintes. Dans ce cas, nous pouvons définir $\prec_{\mathbb{I}}$ comme : $(cs_1, ps_1, ns_1) \prec_{\mathbb{I}} (cs_2, ps_2, ns_2) \equiv (cs_1 + ps_1) > (cs_2 + ps_2)$. Nous représentons ici la capacité qu'aura la première configuration à satisfaire potentiellement plus de contraintes que la deuxième, ayant un plus grand nombre de contraintes satisfaites (certainement et possiblement).
3. privilégier un minimum de contraintes violées, dans ce cas $\prec_{\mathbb{I}}$ est défini comme : $(cs_1, ps_1, ns_1) \prec_{\mathbb{I}} (cs_2, ps_2, ns_2) \equiv ns_1 < ns_2$.

Notons que dans les définitions données ci-dessus $\prec_{\mathbb{I}}$ est bien un ordre total, dans le cas de configurations équivalentes (*e.g.* $cs_1 = cs_2$ dans le premier cas de figure) la première testée sera retenue comme la plus intéressante. Ceci est cohérent puisque les configurations sont équivalentes en termes de satisfaction (certaine, partielle ou insatisfaction) de contraintes, il nous faut un moyen de discrimination, le critère de première évaluation est donc recevable.

#### *3.2.13.4 Un algorithme de recherche locale continue (RLC)*

Nous proposons maintenant l'algorithme 3.7 d'extension de la recherche locale aux domaines continus. Les notions de diversification et d'intensification sont gérées respectivement par deux boucles *tant que* imbriquées. La première assure une exploration « équitable » de l'espace de recherche en effectuant des redémarrages aléatoires (*random restarts*) de l'algorithme, elle peut être assimilée à une recherche en largeur. Au contraire, la boucle d'intensification va concentrer la recherche autour d'une zone prometteuse de l'espace de recherche et est à rapprocher d'une recherche en profondeur.

Notre algorithme de RLC nécessite la définition de quatre paramètres :

1. le nombre de *random restarts* : maxEssais permettant de régler la notion de diversification,
2. le nombre d'itérations associées à chaque point de départ : maxEtapes permettant de régler la notion d'intensification,
3. le nombre de voisins $nb_{\text{vois}}$ générés à chaque itération,
4. le facteur d'échelle $\alpha$, qui permet d'accentuer la notion d'intensification en réduisant l'espace de recherche à chaque pas d'application de l'algorithme.

L'algorithme 3.7 cherche à l'intérieur d'un espace de recherche $\mathbf{B}$ une solution à un ensemble de contraintes $C = \{c_1, \ldots, c_n\}$. Le principe est le suivant :

1. générer aléatoirement une configuration dans l'espace de recherche,

ALG. 3.7 – Recherche Locale Continue (RLC).

```
1  RLC(in: C, P ∈ 𝕀^n, α ∈ ℝ;
       in: maxEssais,maxEtapes,nb_vois ∈ ℤ;
       out: meilleur ∈ 𝕀^n)
2  begin
3   meilleur ← configurationAléatoire(C, P)
4   courant ← meilleur
4   nbEssais ← 0
    % Boucle de diversification
5   while (non estSolution(courant)) ∧ (nbEssais < maxEssais)
6     courant ← configurationAléatoire(C, P)
7     nbEtapes ← 0
      % Boucle d'intensification
8     while (non estSolution(courant)) ∧ (nbEtapes < maxEtapes)
9       V ← voisinage(α, nb_vois, courant, C)
10      courant ← meilleurVoisin(V, C)
11      if (coût(courant) ≺_𝕀 coût(meilleur)) then
12        meilleur ← courant
13      end
14      réduire(α)
15      nbEtapes + +
16    endwhile
17    nbEssais + +
18  endwhile
19  return meilleur
20 end
```

2. constituer le voisinage de cette configuration,
3. évaluer les voisins en termes de satisfaction $\mathcal{CPN}$ des contraintes,
4. sélectionner le meilleur voisin et en faire la configuration courante,
5. sauvegarder la configuration courante si elle est la meilleure rencontrée jusqu'alors (au sens de l'opérateur de comparaison $\prec_{\mathbb{I}}$),
6. retourner à l'étape 2 jusqu'à ce qu'un nombre d'itérations (maxEtapes) soit atteint ou qu'une solution soit trouvée (intensification),
7. retourner à l'étape 1 jusqu'à ce qu'un nombre d'itérations (maxEssais) soit atteint ou qu'une solution soit trouvée (diversification),
8. retourner la meilleure configuration.

#### *3.2.13.5 Une métaheuristique continue : la Recherche Taboue Continue (RTC)*

Comme nous l'avons présenté plus haut, le mécanisme tabou est une métaheuristique permettant de sortir des optima locaux. Elle consiste à ajouter les configurations récemment visitées à une liste taboue $\mathcal{T}$ pour un nombre $\mathtt{t}$ d'itérations. Dans le cas discret, sa mise en œuvre est donc basée sur un test d'égalité des configurations : une configuration est taboue (ne peut être choisie) si elle appartient à la liste taboue (c'est-à-dire si elle est égale à une configuration présente dans $\mathcal{T}$). Deux problèmes majeurs apparaissent lors de l'adaptation de ce mécanisme aux domaines continus :

1. nous devons être en mesure de tester l'égalité de deux configurations continues,
2. l'espace de définition des configurations étant bien trop vaste, la probabilité de retomber deux fois sur exactement la même configuration est proche de zéro.

Afin de résoudre ces problèmes, nous proposons de marquer comme tabou non plus une configuration (c'est-à-dire un point de l'espace de recherche), mais une zone autour de cette configuration. Ceci peut être mis en place de façon très simple pour les intervalles en ne considérant plus l'égalité entre deux configurations, mais l'intersection de leurs domaines par exemple. Une configuration est considérée taboue si un domaine de définition d'une de ses variables intersecte un domaine de définition des variables d'une configuration de la liste taboue. Celle-ci est donc interdite pour un nombre $\mathtt{t}$ d'itérations et la recherche se déplace alors vers des zones non encore explorées.

#### *3.2.13.6 Conclusion* MAX-NCSP*/ RL*

Nous venons de présenter le cadre des MAX-NCSP, ainsi que deux algorithmes permettant de calculer l'approximation intérieure d'un problème de ce type. Un de ces algorithmes nécessite de trouver rapidement un « bon » point de départ en termes de satisfaction de contraintes. Pour ce faire, nous avons suggéré l'utilisation d'une méthode de recherche locale et avons donc proposé l'adaptation aux domaines continus du cadre formel de la recherche locale ainsi que la définition d'une métaheuristique continue : la recherche taboue continue (RTC).

L'approche de résolution de problèmes MAX-NCSP présentée est générique et permet la résolution de tout problème appartenant à la classe des MAX-NCSP. Toutefois, dans le cadre de cette thèse, nous nous intéressons aux problèmes de placement de caméra, c'est donc à cette classe de problèmes que nous allons nous intéresser.

La section suivante est consacrée à l'instanciation de ces algorithmes et de ces cadres formels à un problème de placement de caméra en environnement virtuel. En effet, comme nous l'avons présenté en introduction, l'approche MAX-NCSP semble particulièrement adaptée à cette classe de problèmes.

## 3.3 Une approche MAX-NCSP pour le problème de placement de caméra en environnement 3D

Afin de résoudre un problème de placement de caméra en tant que MAX-NCSP, nous devons instancier les algorithmes présentés en section 3.2.10.2, ainsi que le cadre de la recherche locale introduit en section 3.2.11 afin de prendre en compte les spécificités liées au contrôle de caméra. Ensuite, nous présentons l'expression des propriétés cinématographiques en tant que contraintes numériques. Ces expressions permettent la résolution d'un problème de placement de caméra par notre algorithme MAX-NCSP.

### 3.3.1 Instanciation aux intervalles du cadre RLC

L'algorithme de résolution employé pour résoudre les problèmes de placement de caméra en tant que MAX-NCSP est celui présenté par l'algorithme 3.6, pour lequel la méthode configurationAléatoire est instanciée par une procédure de recherche locale continue définie sur les intervalles. Pour plus d'informations sur l'implémentation de la recherche locale appliquée aux intervalles, le lecteur est invité à se référer à l'annexe C.

#### *3.3.1.1 Une configuration d'intervalles*

Conformément à la définition d'une configuration continue (cf. section 3.2.13.1), une configuration d'intervalles est définie comme :

$$\langle d_1 \times \cdots \times d_n \rangle | \forall i \in [1..n], d_i \subseteq D_i \in \mathbb{I}^n$$

Une configuration d'intervalles correspond à un produit Cartésien d'intervalles à bornes flottantes.

#### *3.3.1.2 Un voisinage d'intervalles*

Deux fonctions $\text{voisinage}_1$ et $\text{voisinage}_2$ ont été définies afin de tester l'impact de la notion de voisinage sur les performances de l'algorithme. La première fonction correspond au voisinage de variable décrit dans la section 3.2.13.2, c'est-à-dire que nous modifions le domaine d'une seule variable de la configuration courante. La fonction $\text{voisinage}_2$ correspond quant à elle a une modification de toutes les variables à la fois. Intuitivement, la première modifie donc un intervalle correspondant à une dimension (*i.e.* une variable) de la configuration courante, alors que la seconde va modifier toutes les dimensions à la fois (*i.e.* tous les intervalles), générant un hyper-pavé englobant la configuration courante.

La fonction échantillonVoisinage est paramétrée par le nombre $nb_{vois}$ de voisins qu'elle doit générer dans le voisinage $\mathcal{V}ois$ construit par les fonctions $\text{voisinage}_1$ ou $\text{voisinage}_2$. Nous devons toutefois prendre garde à explorer $\mathcal{V}ois$ aussi équitablement que possible. Ici encore, deux alternatives s'offrent à nous :

- un premier choix consiste à simplement générer $nb_{vois}$ configurations aléatoirement dans $\mathcal{V}ois$, en utilisant les fonctions de génération pseudo-aléatoire fournie par les langages de haut niveau.
- une deuxième option repose sur une découpe des domaines de $\mathcal{V}ois$ en $nb_{vois}$ parties de taille égale et de choisir un représentant dans chacune de ces zones.

#### *3.3.1.3 Une évaluation par intervalles*

Les fonctions de pénalité sont construites en remplaçant les opérateurs usuels sur les réels par leurs extensions aux intervalles (cf. section 3.2.2.2). De cette façon, nous sommes en mesure de savoir si une contrainte est certainement, partiellement ou non satisfaite. L'évaluation des configurations réside donc dans la comparaison des trois valeurs $cs, ps$ et $ns$ pour chacune des boîtes étudiées et donc dans la définition de l'opérateur $\prec_{\mathbb{I}}$ présenté en section 3.2.13.3.

Afin de tester l'impact de l'évaluation des configurations sur les résultats finaux, nous avons procédé à différentes implémentations de l'opérateur $\prec_{\mathbb{I}}$, à savoir :

- une comparaison stricte des valeurs $cs$ des configurations, dans le respect de l'heuristique *Min-Conflict*, *i.e.* plus la boîte possède de contraintes satisfaites, plus celle-ci est intéressante,
- une comparaison des valeurs $ps$, dans le but de favoriser les boîtes maximisant les contraintes potentiellement satisfaites,
- la prise en compte des contraintes à la fois certainement et possiblement satisfaites, *i.e.* $|cs + ps|$, dans le but de nous rapprocher des zones prometteuses de l'espace de recherche, c'est-à-dire celles maximisant le nombre $|cs + ps|$.

Lors de l'implémentation de notre méthode de Recherche Locale Continue, nous avons testé plusieurs autres opérateurs, en particulier nous avons proposé l'utilisation d'intervalles pour l'évaluation des configurations. Toutefois, les travaux menés concernant la combinaison de tous les opérateurs implémentés, ainsi que l'étude de leurs impacts lors de l'application de la méthode de recherche locale se sont avérés trop immatures pour être présentés dans cette thèse. C'est pourquoi dans ce manuscrit, nous nous limitons volontairement au sous-ensemble très restreint présenté ci-ddessu, des différents évaluateurs implémentés,

Le choix de l'opérateur de comparaison $\prec_{\mathbb{I}}$ n'est pas anodin. La prise en compte des contraintes partiellement satisfaites entraîne une approximation extérieure des solutions du problème MAX-NCSP, *i.e.* nous gardons des zones non solutions mais ne perdons pas de solutions. Au contraire, si nous les omettons, nous guidons la recherche vers une approximation intérieure du problème (nous éliminons des valeurs non solutions et ne gardons que des solutions avérées).

#### *3.3.1.4 Un mécanisme tabou par intervalles*

La dernière étape d'instanciation du modèle RLC aux intervalles consiste en l'implémentation de la métaheuristique taboue. Nous calculons l'intersection des domaines d'une configuration $c$ avec l'ensemble des configurations $t_i$ appartenant à la liste taboue $\mathcal{T}$. Si la taille maximale d'une de ces intersections est supérieure à un seuil déterminé par l'utilisateur, $c$ est alors considérée comme taboue.

L'algorithme 3.8 présente le test d'appartenance d'une configuration $c$ à la liste taboue $\mathcal{T}$ en fonction d'une ensemble de contraintes $C$ et d'une valeur $s$ représentant un seuil minimal de recouvrement.

Cet algorithme est utilisé dans une méthode élimineTaboues qui élimine du voisinage courant toutes les configurations considérées comme taboues. Cette fonction calcule toutes les configurations du voisinage courant qui ne sont pas considérées comme taboues et donc éligibles au statut de prochaine configuration courante de l'algorithme.

#### *3.3.1.5 Un algorithme de RLC adapté aux intervalles*

L'algorithme 3.9 présente l'instanciation d'une méthode de recherche taboue continue adaptée aux intervalles.

ALG. 3.8 – Calcul de l'appartenance d'une configuration à la liste taboue.

```
1  appartenance𝒯(in: config, 𝒯, C, seuil;
                  out: appartient ∈ {0, 1})
2  begin
3   appartient ← 0
4   forall t_i ∈ 𝒯
5      chevauchement ← 1
6      forall v_i ∈ Var(C)
7         I_c ← Domaine(v_i, config)
8         I_{t_i} ← Domaine(v_i, t_i)
9         I_∩ ← ∩(I_c, I_{t_i})
10        ratio ← Largeur(I_∩)/Largeur(I_c)
11        chevauchement ← (chevauchement ∧ (ratio > seuil))
12     endforall
13     appartient ← chevauchement
14  endforall
15  return appartient
16 end
```

ALG. 3.9 – Recherche Taboue Continue adaptée aux Intervalles (RTCI).

```
1  RTCI(in: C, P ∈ 𝕀^n, α ∈ ℝ;
         in: maxEssais,maxEtapes,nb_vois ∈ ℤ;
         out: meilleur ∈ 𝕀^n)
2  begin
3   nbEssais ← 0
4   meilleur ← ∅
5   𝒯 ← ∅
    % Boucle de diversification
6   repeat
7     courant ← configurationAléatoire(C, P)
8     nbEtapes ← 0
      % Boucle d'intensification
9     repeat
10      𝒱_1 ← voisinage(courant, C)
11      𝒱_2 ← échantillonVoisinage(𝒱_1, α, nb_vois)
12      𝒱_3 ← élimineTaboues(𝒯, 𝒱_2)
13      courant ← meilleurVoisin(𝒱_3, C)
14      if (coût(courant) ≺_𝕀 coût(meilleur)) then
15        meilleur ← courant
16      end
17      𝒯 ← (courant, t) ∪ actualiseTaboue(𝒯)
18      réduire(α)
19      nbEtapes + +
20    until (estSolution(courant)) ∨ (nbEtapes > maxEtapes)
21    nbEssais + +
22  until (estSolution(courant)) ∨ (nbEssais > maxEssais)
23  return meilleur
24 end
```

Certaines méthodes utilisées dans cet algorithme n'ont pas été présentées en détails du fait de leur simplicité. Nous allons tout de même décrire leur fonctionnement succinctement :

- configurationAléatoire génère une configuration aléatoire dans l'espace de recherche passé en paramètre,
- coût permet d'obtenir l'évaluation d'une configuration,
- meilleurVoisin retourne le meilleur voisin en termes d'évaluation par rapport à la fonction coût et par rapport à l'opérateur $\prec_{\mathbb{I}}$ d'un voisinage fourni en paramètre,
- actualiseTaboue met à jour le nombre d'itérations $\mathtt{t}$ pour lesquelles une configuration est considérée comme taboue,
- réduire diminue la valeur de $\alpha$ afin d'intensifier la recherche dans une zone prometteuse de l'espace de recherche au cours de la phase d'intensification.

Une fois présenté l'algorithme général utilisé pour résoudre un problème MAX-NCSP, intéressons nous à son instanciation pour la résolution de problèmes de placement de caméra en environnements virtuels. La sous-section suivante est consacrée à la présentation de l'adaptation des propriétés cinématographiques en contraintes numériques.

## 3.3.2 Des propriétés aux contraintes

La formalisation d'un problème de placement de caméra en tant que MAX-NCSP nécessite la traduction des propriétés visuelles spécifiées sur le résultat souhaité à l'écran ou bien les propriétés définies directement sur les paramètres de la caméra en un ensemble de contraintes exploitables par les algorithmes présentés précédemment.

Cette section est donc consacrée à la présentation de la traduction en contraintes d'un certain nombre de propriétés cinématographiques. La section 4.2 du chapitre suivant est dédiée à la présentation détaillée d'un certain nombre de propriétés cinématographiques, le lecteur est invité à se référer à cette section pour plus de précisions sur ce type de propriétés. Pour chacune des contraintes spécifiées dans la suite de cette section, les représentations de la caméra et des objets sont celles présentées en annexe A.

### *3.3.2.1 La propriété de cadrage*

La propriété de cadrage d'un objet consiste à contraindre sa projection dans un rectangle à l'image. La représentation d'un objet (cf. section A.0.2) consiste en une sphère englobante, c'est-à-dire une position 3D $P$ et un rayon $r$. Le cadre $C$ est quant à lui défini en tant que rectangle dans l'image par quatre composantes $(X_{min}, X_{max}, Y_{min}, Y_{max})$.

La propriété de cadrage est vérifiée si la projection de la sphère englobante de l'objet d'intérêt est contenue dans $C$. Soit $P_{cam}(x_{cam}, y_{cam}, z_{cam})$ le transformé de $P(x, y, z)$ dans le repère de la caméra (les calculs de changement de repère et de projection d'un point 3D sont décrits en détails dans la section 4.5.3.1), la contrainte de cadrage se définit alors comme suit (où $\gamma_C$ représente la distance focale de la caméra) :

$$\begin{cases} Xmin & \leq \big(x_{cam} - r\big)/\big(z_{cam}/\gamma_C\big) \\ Xmax & \geq \big(x_{cam} + r\big)/\big(z_{cam}/\gamma_C\big) \\ Ymin & \leq \big(y_{cam} - r\big)/\big(z_{cam}/\gamma_C\big) \\ Ymax & \geq \big(y_{cam} + r\big)/\big(z_{cam}/\gamma_C\big) \end{cases}$$

#### *3.3.2.2 La propriété d'orientation*

À partir des représentations de la caméra et des objets pris en compte (cf. annexe A), la contrainte d'orientation d'un objet à l'écran est basée sur le produit scalaire entre le vecteur vision de la caméra et le vecteur orientation de l'objet.

L'objectif est de minimiser l'écart entre l'orientation souhaité pour l'objet et le vecteur reliant la caméra et l'objet en question. Afin de ne pas être trop restrictif, nous ajoutons un paramètre permettant de définir un angle entre l'écart idéal entre l'orientation de l'objet et du vecteur objet-caméra. Ce paramètre $\alpha$ représente ainsi un intervalle de valeurs acceptées comme satisfaisant la propriété d'orientation.

Soit $C(x_C, y_C, z_C)$ la position de la caméra, $O(x_O, y_O, z_O)$ la position de l'objet et $\vec{V}(x_V, y_V, z_V)$ le vecteur unitaire représentant l'orientation de ce dernier, la contrainte $c$ associée à la propriété d'orientation est la suivante :

$$\begin{aligned} x_T &= x_C - x_O \\ y_T &= y_C - y_O \\ z_T &= z_C - z_O \\ n_T &= \sqrt{{x_T}^2 + {y_T}^2 + {z_T}^2} \\ \vec{T} &= (\frac{x_T}{n_T}, \frac{y_T}{n_T}, \frac{z_T}{n_T}) \end{aligned}$$

$$c = (\vec{T}\vec{N} \leq \alpha)$$

Le principe consiste à calculer le vecteur unitaire $\vec{T}$ reliant la caméra à l'objet d'intéret, puis à calculer son produit scalaire avec l'orientation intrinsèque de l'objet. Enfin, comme nous l'avons présenté plus haut, un angle $\alpha$ permet d'ajouter de la flexibilité à la satisfaction de la propriété.

#### *3.3.2.3 La propriété de distance entre la caméra et l'objet*

Cette propriété permet de contraindre la distance entre la caméra et un objet d'intérêt dans la scène 3D, en assurant que cette distance soit supérieure à une valeur définie par l'utilisateur. La contrainte associée est spécifiée en fonction de la position de l'objet d'intérêt $O(x_O, y_O, z_O)$, de celle de la caméra $C(x_C, y_C, z_C)$ et d'une distance définie par l'utilisateur $d$ :

$$\sqrt{(x_C - x_O)^2 + (y_C - y_O)^2 + (z_C - z_O)^2} \geq d$$

#### *3.3.2.4 La propriété « gardien »*

Cette propriété permet de modéliser une caméra possédant un champ de vue ($\gamma$) fixe, blabla2D

## 3.4 Résultats

Les résultats présentés dans cette section sont obtenus à partir de problème ayant attrait au placement de caméra en environnement virtuel. Toutefois, comme nous l'avons précisé tout au long de ce chapitre, tout type de problème MAX-NCSP est pris en compte par approche. Nous proposons deux jeux de tests différents pour illustrer la pertinence de notre approche : le premier consiste en une application typique

de composition visuelle, alors que le second reprend l'idée du problème de gardien de musée, appliquée à un environnement virtuel.

Les tests sont menés sur chacun des 5 algorithmes différents, à savoir :

- l'algorithme naïf de bissection-évaluation,
- l'algorithme naïf amélioré qui étudie seulement les boîte prometteuses,
- l'algorithme amélioré avec évaluation par point central,
- l'algorithme amélioré avec évaluation par recherche locale continue,
- l'algorithme amélioré avec évaluation par recherche locale continue et extension intérieure,
- l'algorithme amélioré avec évaluation par recherche locale continue, extension intérieure et propagation des contraintes possiblement satisfaites.

De plus, chacun des algorithmes est paramétré par les trois évaluateurs suivants :

- une comparaison par rapport au nombre de contraintes certainement satisfaites du pavé courant,
- une comparaison par rapport au nombre de contraintes certainement et possiblement satisfaites du pavé courant,
- une comparaison par rapport au nombre de contraintes non satisfaites du pavé courant.

### 3.4.1 Jeu de test : composition visuelle

La composition visuelle consiste à spécifier le placement d'objets à l'intérieur de l'image résultat. L'utilisateur peut ainsi caractériser l'agencement de l'image résultat sans avoir à manipuler directement la caméra. L'exemple présenté ici va amener à la construction de deux classes de solutions distinctes, afin de mettre en valeur les spécificités de notre méthode.

La scène 3D prise en compte comporte trois objets distincts $A$, $B$ et $C$ alignés blabla. La description du problème est basée sur l'utilisation de deux types de propriétés :

- la propriété de *framing* : qui permet de spécifier l'appartenance d'un objet à l'intéreiur d'un rectangle 2D dans l'image finale,
- la propriété d'orientation : qui permet de spécifier sous quel angle de vue (face, profil, etc.) un objet doit être filmé.

Le problème est composé de cinq propriétés distinctes définies sur les trois objets $A$, $B$ et $C$ :

1. cadrer l'objet $A$ à gauche de l'image résultat, dans le cadre F1,
2. cadrer l'objet $B$ à droite de l'image résultat, dans le cadre F2,
3. cadrer l'objet $C$ à droite de l'image résultat, dans la même *frame* que l'objet $B$ (*i.e.* F2),
4. voir l'objet $A$ de face,
5. voir l'objet $C$ de face.

Afin d'illustrer les différents algorithmes mis en œuvre dans ce chapitre, ainsi que les différents évaluateurs, nous avons résolu ce problème en paramétrant notre algorithme afin qu'il teste toutes les configrations possibles, tant au niveaude l'algorithme de résolution utilisé, que des évaluateurs des configurations.

### 3.4.2 Jeu de test : problème de gardien de musée

Ce deuxième jeu de test reproduit une version « orientée caméra » du problème de gardien de musée. Le problème consiste en la génération aléatoire d'objets 3D dans une scène, et à la définition de propriétés de visibilité entre des caméras virtuelles et ces objets. Le problème consiste donc à générer des positions

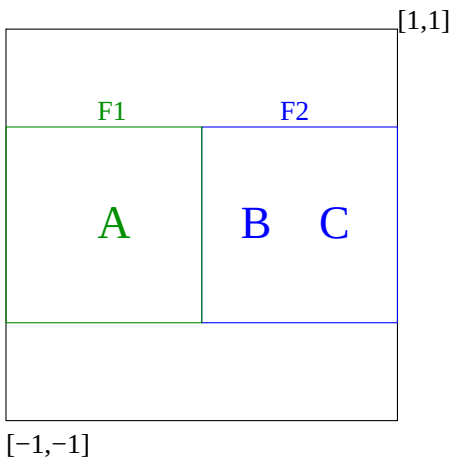



Figure 3.13 – Cadres mis en jeu dans le jeu de test de composition visuelle. Le cadre F1 doit contenir l'objet $A$, alors que $B$ et $C$ doivent appartenir à F2.

de caméras maximisant le nombre d'objets vus et pouvant être ainsi « surveillés ». Ainsi, la résolution du problème va amener à la construction de différentes configurations de caméra, satisfaisant chacune un sous-ensemble de propriétés.

## 3.5 Conclusion

Nous avons proposé dans ce chapitre un algorithme générique de résolution de problèmes de maximisation de satisfaction d'inéquations numériques. Notre approche MAX-NCSP permet de résoudre indifféremment des problèmes sur-contraints, sous-contraints ou bien-contraints composés d'inégalités numériques. En effet, notre méthode consiste à calculer les pavés intérieurs au *store* de contraintes maximisant le nombre de contraintes satisfaites et à fournir la liste des contraintes satisfaites à l'utilisateur dans le cas de résolution d'un problème sur-contraint. Nous assurons également la propriété de complétude en caractérisant toutes les boîtes de l'espace de recherche initial par rapport à un triplet de valeurs $\langle cs, ps, ns\rangle$ représentant respectivement les ensembles de contraintes certainement satisfaites, possiblement satisfaites et non satisfaites du *store* de contraintes. Cette caractérisation $\mathcal{CPN}$ de l'ensemble des pavés de l'espace de recherche permet d'éliminer rapidement les boîtes inintéressantes lors du processus de résolution. En effet, en conservant tout au long de la résolution une valeur $m$ représentant l'optimum courant du nombre de contraintes certainement satisfaites, nous pouvons nous consacrer à l'étude des pavés pouvant amener à une amélioration de la valeur $m$, c'est-à-dire lorsque les ensembles $\mathcal{C}$ et $\mathcal{P}$ de la boîte étudiée contiennent plus de contraintes que l'optimum courant, *i.e.* : $|\mathcal{C}| + |\mathcal{P}| \geq m$.

L'algorithme basique consiste en trois étapes tissées successivement :

1. la recherche d'un point de départ intérieur (*i.e.* solution à un sous-ensemble des inéquations du système de contraintes).
2. une étape d'extension intérieure permettant de maximiser la zone de l'espace de recherche située dans la proximité du point de départ obtenu à l'étape précédente.

3. une étape de propagation de contraintes appliquée à l'extension intérieure obtenue précédemment. La propagation concerne uniquement l'ensemble des contraintes $\mathcal{P}$ossiblement satisfaites, dans le but de restreindre l'espace de recherche à des zones potentiellement intéressantes pour l'étape suivante de l'algorithme. En effet, la prise en compte des seules contraintes possiblement satisfaites permet de favoriser la caractérisation des contraintes du problème entre $\mathcal{C}$ertainement et $\mathcal{N}$on satisfaites, ainsi l'algorithme converge plus rapidement.

Notre algorithme est original dans la mesure où il est le seul, à notre connaissance, qui calcule l'ensemble des pavés satisfaisant certainement le maximum de contraintes d'un problème MAX-NCSP : $\mathcal{S} = \langle C, \mathcal{V}, \mathbf{D} \rangle$ et ce quelque soit la nature de celui-ci. En effet, si $\mathcal{S}$ est bien-contraint ou sous-contraint, l'algorithme calcul l'ensemble des pavés satisfaisant certainement l'ensemble des contraintes, *i.e.* tous les pavés $\mathbf{B}$ tels que : $|\mathsf{CS}(C, \mathbf{B})| = |C|$. Au contraire, si $\mathcal{S}$ est sur-contraint, l'algorithme calcule l'ensemble des zones de l'espace de recherche maximisant le nombre de contraintes satisfaites. Ces dernières sont présentées à l'utilisateur, conjointement à la liste des contraintes satisfaites dans chacune des zones max-sat.

Nous avons développé notre méthode MAX-NCSP afin de proposer une approche de résolution numérique pour les problèmes de placement de caméra en environnements virtuels. En effet, les travaux existants proposant des approches numériques pour répondre aux problèmes de placement de caméra sont généralement inadaptés à la prise en compte de CSP sur-contraints, alors que la spécification d'un problème de placement de caméra sous formes de contraintes numériques conduit souvent à la création de CSPs sur-contraints. Comme nous l'avons mentionné, notre approche est basée sur la mise à jour d'une valeur courante représentant l'optimum du nombre de contraintes satisfaites lors du processus de résolution présenté ci-avant. Ainsi, les performances globales de l'algorithme sont très dépendantes de l'étape de découverte du pavé maximisant le nombre de contraintes satisfaites. Afin d'identifier cette boîte le plus tôt possible, nous avons proposé de développer une méthode de recherche locale adaptée aux domaines continus nous permettant d'instancier la procédure de recherche d'un point de départ pour l'étape d'extension intérieure. Les méthodes de recherche locale sont en effet réputées pour être particulièrement efficaces lors de la résolution de CSPs discrets. En instanciant ce cadre de méthode de résolution incomplète, nous souhaitons déterminer le plus tôt possible la boîte maximisant le nombre de contraintes du problème.

Notre définition d'une méthode de recherche locale continue (RTC), c'est-à-dire adaptée à des variables représentées par des intervalles à bornes flottantes, consiste en l'instanciation des concepts de base du cadre de la recherche locale, à savoir les notions de :

- configuration,
- fonction de pénalité,
- voisinage,
- évaluation ou comparaison.

De même, afin d'améliorer les performances de notre algorithme RTC de recherche locale continue, nous avons proposé une instanciation de la métaheuristique classique taboue permettant d'éviter les problèmes d'optimum locaux dans les algorithmes de recherche locale.

Une implémentation de notre cadre de recherche locale continue avec métaheuristique taboue a été proposée et intégrée dans la méthode MAX-NCSP. La pertinence de notre approche est illustrée dans une série d'exemples. Toutefois, notre approche souffre de la limitation de son instanciation aux problèmes amenant à l'obtention de continuums de solutions, c'est-à-dire à des systèmes d'inéquations numériques. De même, le cadre de recherche locale continu présenté consiste en un travail préliminaire et nécessite une étude concernant les opérateurs d'agrégation des fonctions de coût et des opérateurs liés à la métaheuristique taboue continue.

# CHAPITRE 4

# Les volumes sémantiques


« Ceci n'est pas une pipe. »
René MAGRITTE - La Trahison des images - 1929.
Los Angeles, County Museum.

Dans le chapitre précédent nous avons proposé une réponse purement numérique au problème de placement de caméra virtuelle. Comme nous l'avons montré, les solutions obtenues répondent *le mieux possible* au problème posé par l'utilisateur, c'est-à-dire en maximisant le nombre de propriétés satisfaites du problème. Toutefois, le résultat de notre algorithme MAX-NCSP consiste *simplement* en une instanciation des paramètres de la caméra. Cette approche ne permet donc pas de calculer plusieurs solutions lorsque la description utilisateur donne naissance à différentes classes de solutions équivalentes. L'identification et la caractérisation de ces configurations numériquement équivalentes (*i.e.* solutions du problème) mais sémantiquement différentes (*i.e.* répondant au problème de manière distinctives) est impossible par application de l'algorithme MAX-NCSP. En effet, les méthodes numériques usuelles de résolution utilisées (propagation de contraintes et recherche locale) ne permettent pas de conserver l'aspect sémantique des propriétés au cours de la résolution du problème. Dans ce chapitre, nous proposons une approche dédiée à l'identification et à la classification des solutions d'un problème de placement de caméra : la *méthode des volumes sémantiques* ou plus simplement *volumes sémantiques*.

## 4.1 Introduction

L'approche *orientée propriété* que nous proposons pour le problème de composition visuelle est basée sur la méthode de partition spatiale BSP (*Binary Space Partitioning*) proposée par Fuchs, Kedem et Naylor [FKN79, FKN80] et utilisée en informatique graphique. Cette méthode consiste à subdiviser les objets de l'espace en les découpant successivement à partir de plans identifiés dans la scène 3D (cf. figure 4.1). Cette subdivision est effectuée jusqu'à ce qu'un critère d'arrêt soit atteint, ce dernier étant spécifié en fonction de l'application visée.

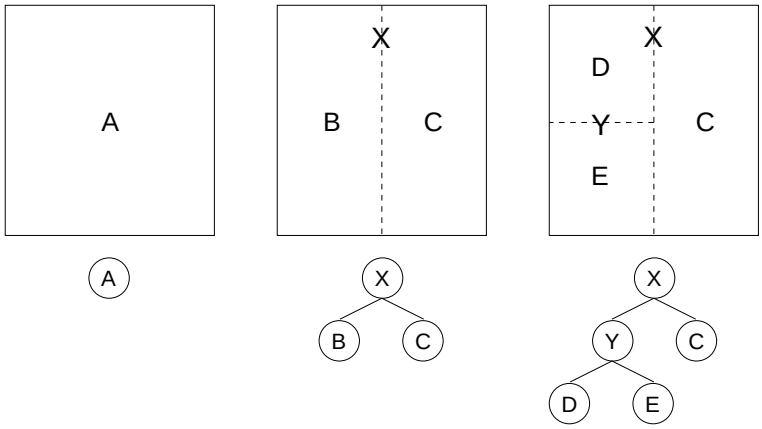



Figure 4.1 – Arbre BSP de découpe d'un polygone suivant l'axe X puis l'axe Y.

En imagerie numérique, la méthode BSP peut être utilisée à la fois pour améliorer l'efficacité du rendu d'une scène et pour améliorer l'efficacité du calcul de collisions. Dans le premier cas, le critère d'arrêt de la subdivision est la convexité d'un polygone (c'est-à-dire que l'on subdivise les objets jusqu'à ce que chaque partie soit convexe), dans le deuxième cas il concerne la *simplicité* des objets de la scène par rapport aux tests de collisions. L'application de la méthode BSP permet la création d'un arbre qui catégorise les polygones d'une scène par rapport à une subdivision de l'espace. Cette structure de données permet une élimination très rapide des polygones lors de l'exploration d'une scène 3D à décor fixe (les polygones du décor appartiennent à l'arbre BSP). En effet, une simple étude de la position de l'explorateur dans le monde permet d'éliminer tous les polygones ne pouvant être vus depuis cette position. Un arbre BSP permet d'améliorer l'efficacité du rendu lors de l'exploration d'une scène 3D à décor fixe (dans un jeu vidéo par exemple) de manière significative.

Concernant la détection de collisions (en robotique par exemple), la méthode BSP permet de découper les objets complexes en fragments plus simples pour lesquels un algorithme de détection de collisions efficace peut être employé.

Dans nos travaux, nous proposons de découper successivement l'espace de recherche des positions de caméra. Cette technique est à rapprocher des travaux de Koenderink et van Doorn [KvD79] concernant les *aspects visuels* et de Plantinga et Dyer [PD90] sur le partitionnement de l'espace en fonction du point de vue (*viewpoint space partitioning*) dans le domaine de la reconnaissance d'objets.

L'idée sous-jacente aux aspects visuels est de réunir dans un même ensemble de positions 3D tous les points de vue d'un unique polyèdre menant à une image ayant les mêmes propriétés topologiques par rapport à celui-ci. Les frontières des espaces ainsi obtenus sont définies lorsqu'une modification du point de vue amène à un changement d'apparence du polyèdre dans l'image résultat, cf. figure 4.2.

Une fois toutes les frontières calculées pour un même objet, il est possible de recréer des régions consistantes de l'espace par rapport à l'aspect de l'objet à l'écran, ce que les auteurs ont appelé les partitions de l'espace par point de vue (ou *viewpoint space partitions*).

Nous proposons une extension des aspects visuels et des partitions de l'espace par points de vue selon deux axes :

1. la prise en compte d'objets multiples d'une scène,

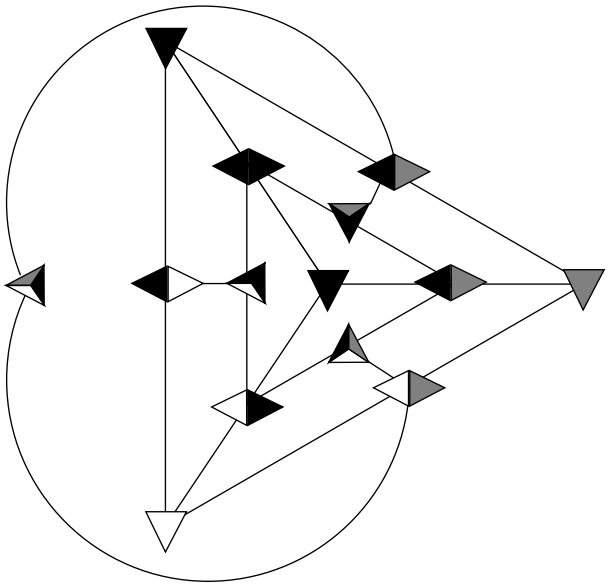


Figure 4.2 – Graphe d'aspects d'un tétraèdre, d'après Koenderink et van Doorn [KvD79]. On distingue trois types d'aspects suivant le nombre de faces visibles à partir d'un point de vue.

2. la prise en compte de caractéristiques cinématographiques (*e.g.* l'occlusion entre objets, les angles de vue relatifs à un objet, etc.) plutôt que topologiques.

Conformément aux *viewpoint space partitions*, nous introduisons la notion de *volume sémantique*, comme étant un volume des positions possibles d'une caméra virtuelle qui aboutissent à des rendus qualitativement équivalents en termes de propriétés cinématographiques, c'est-à-dire aboutissant à des prises de vue *sémantiquement* équivalentes.

La figure 4.3 illustre deux prises de vue sémantiquement équivalentes issues d'un volume sémantique dont la description associée est : « Obtenir une vue de profil de Super Calvin ». Cette description ne spécifiant pas quel profil est visé, les deux vues sont donc possibles. Chaque volume sémantique est accompagné d'une liste d'*étiquettes sémantiques* permettant la caractérisation de ce volume en fonction des propriétés satisfaites. En effet, un même volume peut correspondre à plusieurs propriétés cinématographiques à la fois. Ces étiquettes sont identifiées à partir de la littérature cinématographique [Ari76, Mas65, Kat91] et concernent soit (*i*) un unique objet, *e.g.* pour la propriété d'angle de vue, soit (*ii*) un couple d'objets (par exemple pour les propriétés d'occlusion ou de positionnement relatif à l'écran). Il est donc possible de découper l'espace 3D des positions de caméra pour chaque objet et pour chaque couple d'objets de la scène.

Les différentes étapes de création et d'exploitation des volumes sémantiques sont résumées dans le schéma 4.4 qui présente l'approche globale en séparant le niveau d'interaction utilisateur du niveau numérique.

La procédure suivie pour la création des volumes sémantiques consiste à extraire les propriétés (cf. section 4.2) spécifiées sur les objets composant la scène 3D d'intérêt à partir d'une description effectuée par un utilisateur. La prise en compte de chacune d'elles aboutit à la création d'un ou de plusieurs volumes sémantiques satisfaisant la propriété en question. Une fois tous les volumes créés, nous devons

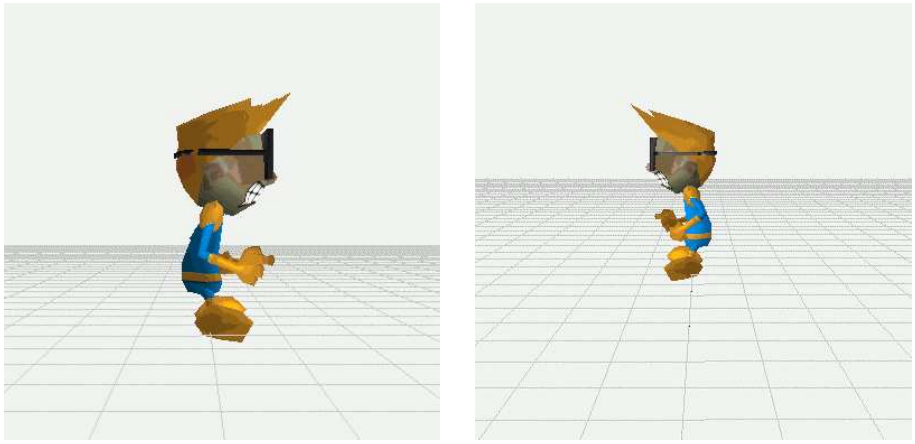


Figure 4.3 – Deux images sémantiquement équivalentes correspondantes à la description « Voir le profil de Super Calvin ».

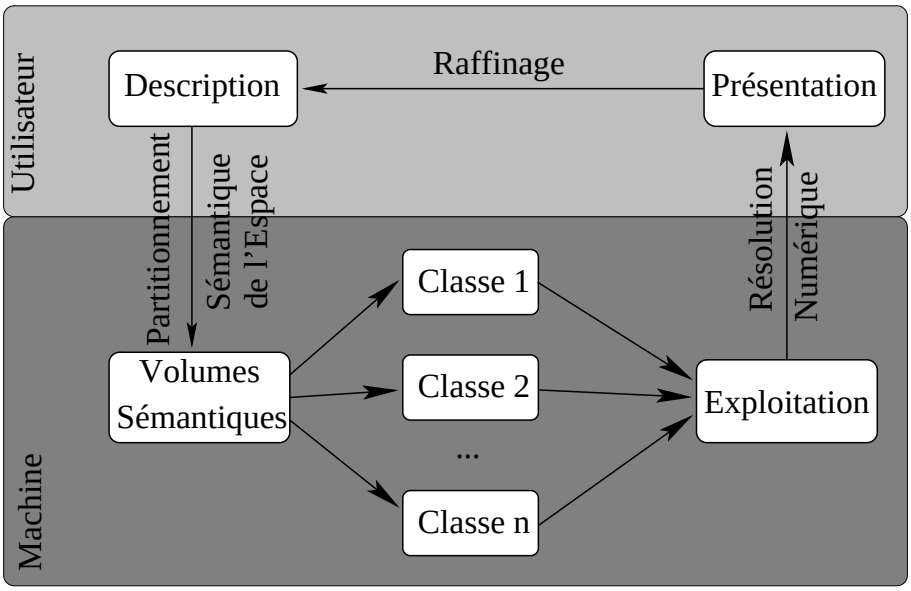



Figure 4.4 – Schéma global de l'approche des volumes sémantiques.

réaliser l'intersection de ceux-ci pour vérifier qu'une (ou plusieurs) zone(s) de l'espace de recherche initial correspond(ent) à la description souhaitée par le réalisateur virtuel. Enfin, nous devons calculer des configurations de caméra représentatives de chacun des volumes, *i.e.* de chaque classe de solutions, afin de les présenter à l'utilisateur.

Comme nous venons de le signaler, le processus de résolution lié aux volumes sémantiques se divise en quatre étapes :

1. la description du problème par l'utilisateur *via* des propriétés cinématographiques sur les objets de la scène 3D,
2. la création et l'intersection des volumes sémantiques répondant au problème,
3. le calcul des meilleurs candidats pour chacun des volumes sémantiques solutions,
4. la présentation des résultats à l'utilisateur.

Dans la suite de ce chapitre, nous formalisons tout d'abord la notion de *volume sémantique*, puis nous détaillons les différentes propriétés prises en compte avant de décrire le processus de résolution numérique. Enfin, avant de conclure, nous présentons des jeux d'essais illustrant les caractéristiques de notre approche.

### 4.1.1 Un volume sémantique

Les volumes sémantiques sont une extension des aspects visuels à la prise en compte de plusieurs objets et en fonction d'un ensemble de propriétés cinématographiques plutôt que topologiques. Un volume sémantique contient l'ensemble des positions de caméra aboutissant à la satisfaction du même ensemble de propriétés, c'est-à-dire possédant les mêmes caractéristiques à l'écran. Nous formalisons la notion de volume sémantique de la manière suivante :

**Définition 4.1** (volume sémantique)**.** Un volume sémantique $v_s$ est défini par un couple :

$$v_s = \langle \mathcal{S}, \mathcal{V} \rangle$$

où $\mathcal{S}$ représente une conjonction d'étiquettes sémantiques, et $\mathcal{V}$ est un sous-ensemble de $\mathbb{R}^3$.

Les étiquettes sémantiques correspondent aux propriétés satisfaites à l'intérieur du volume $\mathcal{V}$. Celles-ci sont présentées en détails dans la section 4.2 et peuvent correspondre par exemple à l'étiquette sémantique suivante : $\{\mathsf{ProfilGauche(A)} \wedge \mathsf{PlanAméricain(B)} \wedge \mathsf{Occlusion(A,B)}\}$.

### 4.1.2 Opérateur de filtrage géométrique $G_f$

Le volume $\mathcal{V}$, associé au volume sémantique $v_s = \langle \mathcal{S}, \mathcal{V} \rangle$, englobe les positions de caméra pour lesquelles les propriétés $\mathcal{S}$ sont *possiblement* satisfaites. Ainsi, une caméra positionnée à l'intérieur du volume $\mathcal{V}$ peut satisfaire la propriété $p$, sous réserve d'orientation correcte. Au contraire, toute configuration de caméra située en dehors de ce volume ne peut aboutir à la satisfaction de $p$ et ce quelque soit l'orientation choisie. La propriété de correction est donc assurée par le principe même de construction des volumes sémantiques.

Toutefois, le calcul d'une orientation étant nécessaire pour assurer la satisfaction complète de la propriété cinématographique, les configurations de caméra situées à l'intérieur d'un volume sémantique correspondent à des *instanciations correctes mais partielles* des paramètres de la caméra. L'instanciation

des paramètres d'orientation de la caméra sera traitée dans la section 4.3.3 consacrée au processus de résolution numérique.

Un volume sémantique correspond à un sous-ensemble de $\mathbb{R}^3$ englobant les configurations partielles et correctes de caméra satisfaisant une propriété utilisateur. Chaque volume est obtenu par application de l'opérateur de filtrage géométrique $G_f$ à un espace de recherche et à une propriété cinématographique. Nous proposons la définition suivante pour l'opérateur de filtrage géométrique $G_f$ :

**Définition 4.2** (opérateur de filtrage géométrique $G_f$)**.** Soit $p$ une propriété cinématographique et $\mathcal{E}$ un espace de recherche. Le volume sémantique $v_s = \langle \mathcal{S}, \mathcal{V} \rangle$ correspondant aux positions correctes de caméra dans $\mathcal{E}$ satisfaisant *possiblement* la propriété $p$ est obtenu par application de l'opérateur $G_f$ :

$$v_s = \langle \mathcal{S}, \mathcal{V} \rangle = G_f(p, \mathcal{E})$$

où $\mathcal{S}$ représente la conjonction des étiquettes sémantiques associées à la propriété $p$ et $\mathcal{V} \subseteq \mathcal{E}$ le volume englobant l'ensemble des positions de caméra satisfaisant possiblement $p$.

L'application de cet opérateur à un espace de recherche et à une propriété peut conduire à l'obtention d'un ensemble de volumes non-connexes une fois les positions de caméra inconsistantes éliminées (*i.e.* $\mathcal{V}$ peut représenter un volume non-connexe). L'opérateur de filtrage géométrique est *complet* dans la mesure où il n'élimine aucune position de caméra pouvant amener à la satisfaction de la propriété $p$ à laquelle il est appliqué.

### 4.1.3 Intersection des volumes sémantiques

Nous avons proposé une définition de la notion de volume sémantique et présenté un opérateur permettant la création d'un volume sémantique en fonction d'un espace de recherche et d'une propriété utilisateur. La description d'un problème de placement de caméra en environnement virtuel consiste en une liste de plusieurs propriétés. Afin de résoudre le problème, nous devons traiter la conjonction de ces propriétés afin d'obtenir la solution.

La solution du problème initial, décrite comme une conjonction de propriétés cinématographiques ($\bigwedge_i p_i$), correspond à une intersection Booléenne des volumes sémantiques ($v_{s_i}$) obtenus par application de l'opérateur $G_f$ à chaque propriété $p_i$ dans l'espace de recherche initial $\mathcal{E}$. Toutefois, les volumes géométriques obtenus par intersections des $\mathcal{V}_i$ contiennent les instanciations partielles de caméra décrites par l'union des étiquettes sémantiques ($\mathcal{S}_i$).

L'approche des volumes sémantiques modélise un problème de placement de caméra par à un ensemble de propriétés cinématographiques ($p_i$) et le résout de la manière suivante :

$$\begin{aligned} \bigcup_i p_i &= \bigcap_i G_f(p_i, \mathcal{E}) \\ &= \bigcap_i \langle \mathcal{S}_i, \mathcal{V}_i \rangle \\ &= \langle \bigwedge_i \mathcal{S}_i, \bigcap_i \mathcal{V}_i \rangle \end{aligned}$$

## 4.2 Propriétés

Cette section présente les différentes propriétés cinématographiques que nous proposons aux utilisateurs dans notre approche de placement de caméra. Celles-ci sont issues d'ouvrages majeurs de la littérature cinématographique tels que « La grammaire du langage filmé » de Daniel Arijon [Ari76], « Film Directing Shot by Shot: Visualizing from Concept to Screen » de Steven Katz [Kat91] ou bien encore

« The Five C's of Cinematography: Motion Picture Filming Technique » de Joseph Mascelli [Mas65]. Ces ouvrages traitent des techniques cinématographiques en général et plus particulièrement des problèmes liés au montage, à la composition, au cadrage ou à la prise de vue. En nous basant sur ces trois ouvrages, nous avons défini un ensemble de propriétés cinématographiques fréquemment utilisées en composition visuelle.

Dans la suite, nous limiterons volontairement le degré de liberté de roulis (*roll*) de la caméra (cf. annexe A sur les modèles de représentation de la caméra et des objets), c'est-à-dire autour du vecteur vision de la caméra à l'intervalle $[-\frac{\pi}{4}, \frac{\pi}{4}]$. De plus, l'angle de tangage (*pitch*) sera quant à lui limité à l'intervalle $[-\frac{\pi}{2}, \frac{\pi}{2}]$ afin d'interdire une caméra ayant « la tête à l'envers ». Nous adoptons ces conventions dans la mesure où les règles de composition visuelle et de montage ne transgressent que très rarement ces conventions.

### 4.2.1 La propriété de projection

La propriété de projection correspond au concept d'*échelle des plans de vue*, c'est-à-dire au rapport entre la taille du cadre de l'image et la taille des personnages ou des objets dans celle-ci. Six gradations distinctes sont traditionnellement identifiées pour la taille d'un objet filmé à l'écran parmi l'infinité de distances possibles entre la caméra et l'objet d'intérêt. Celles-ci sont illustrées en figure 4.5 et identifiées d'apres [Ari76, Chr03, ww5w] comme suit :

- le très gros plan : montre un seul objet, généralement un détail du visage lorsqu'il s'agit d'un personnage (également appelé *insert* lorsqu'il est question d'un objet). Ce type de plan relativement peu utilisé par le passé se développe dans le cinéma actuel. « Il permet une perception de la réalité très différente de par son échelle inhabituelle, et peut permettre de donner l'impression d'une distorsion par rapport à la réalité. Généralement très bref, il sert la progression du récit ou du suspense en attirant l'attention sur un détail dramatiquement frappant. » ([ww5w]). Darren Aronofsky l'utilise à de nombreuses reprises dans son film « *Requiem for a dream* » afin d'attirer l'attention sur un détail particulièrement marquant du film : la réaction de la pupille des personnages du film lors de la prise d'héroïne. Le réalisateur utilise ce type de plan afin de souligner l'effet dévastateur de l'addiction des personnages.
- le gros plan : permet de montrer des détails d'un personnage ou d'un objet (dans ce cas on parlera de *plan de détail*), sur lesquels l'attention doit être attirée. « Il permet de lire directement la vie intérieure d'un personnage, ses émotions, ses réactions les plus intimes. C'est le plan de l'analyse psychologique » [ww5w]. Les réalisateurs de *western spaghetti* usent et abusent de ce genre de plans de vue, en particulier lors des duels entre les différents protagonistes, comme dans la « Saga trilogie du dollar » composée des trois films : « Pour une poignée de dollars », « Et pour quelques dollars de plus » et de « Le bon, la brute et le truand » tous trois réalisés par Sergio Leone.
- le plan rapproché : cadre un personnage au niveau des épaules ou de la poitrine. « Il correspond par excellence aux plans de dialogues » [ww5w].
- le plan moyen ou plan américain : ce plan cadre les personnages à la ceinture et montre principalement ce qu'ils disent et font sans pour autant attirer exagérément l'attention sur un détail précis de leur jeu. C'est pourquoi il est, lui aussi, classiquement utilisé dans les dialogues. « Il accentue l'intimité, permet de lire les réactions psychologiques, le jeu du visage et des épaules » [ww5w].
- le plan plein cadre : montre un ou plusieurs personnages en pied. Il est généralement utilisé pour situer les protagonistes dans le décor de la scène.
- le plan d'ensemble : correspond par excellence au plan d'exposition d'une scène. La totalité du décor et les personnages qui s'y trouvent est filmée. Les plans d'ensemble sont souvent utilisés

comme introduction ou comme conclusion d'un film en situant le cadre de l'œuvre.

La méthode des volumes sémantiques ne s'intéresse qu'à des plans fixes, non à des séquences, ainsi les notions de dialogue citées en exemple dans les descriptions des différents types de plan servent uniquement à des fins illustratives, afin d'illustrer leurs différentes utilisations. Les six types de plan de vue sont donc mis à disposition de l'utilisateur et chacun d'eux est associé une étiquette sémantique, à savoir :

- TrèsGrosPlan(Objet),
- GrosPlan(Objet),
- PlanRapproché(Objet),
- PlanAméricain(Objet),
- PleinCadre(Objet),
- PlanEnsemble(Objet).

Le volume sémantique correspondant à la propriété de projection est donc calculé en fonction d'un objet de la scène et de l'une des six distances de prise de vue cinématographique décrite précédemment.

En fonction du type de plan de vue, de la taille de l'objet (ou d'un englobant) dans la scène 3D et de la taille de l'image résultat (toutes deux connues) ; il est possible de calculer une taille optimale de l'objet dans l'image résultat et d'en déduire la distance à laquelle la caméra doit se trouver par rapport à l'objet grâce à l'équation suivante :

$$distance = (\frac{tailleObjet}{tailleEcran}) \times (\frac{1}{\tan(\frac{fov}{2})}) \tag{4.1}$$

Afin d'ajouter de la flexibilité à notre système et de minimiser les erreurs dûes à l'utilisation de volumes englobants, nous proposons de prendre en compte un intervalle de valeurs acceptables plutôt qu'une valeur unique déterminant la distance à laquelle placer la caméra. Cet intervalle est calculé, à la manière de Bares *et al.* [BMBT00], en encadrant la distance idéale entre la caméra et l'objet visé par un intervalle de valeurs solutions. Les bornes inférieures et supérieures de cet intervalle nous permettent de définir deux distances minimales et maximales autorisées pour obtenir le plan de vue désiré à l'écran.

L'opérateur géométrique associé à cette propriété utilise ces deux valeurs minimales et maximales comme rayons respectivement intérieurs et extérieurs d'une sphère « creuse » centrée autour de l'objet. Celle-ci englobe les positions de caméra permettant d'obtenir le plan de vue désiré. La figure 4.6 présente un exemple d'implémentation de la propriété de projection (pour une présentation détaillée de l'implémentation de cette propriété, le lecteur est invité à se référer à la section 4.5.3.1).

### 4.2.2 Propriété d'orientation

La propriété d'orientation permet au réalisateur virtuel de spécifier un angle de vue sous lequel il souhaite voir un objet de la scène, *e.g.* de face, de profil, de dos, etc. Nous avons choisi d'offrir un panel de dix angles de vue classiques relativement à un objet, à savoir :

- la vue de profil gauche : ProfilGauche(Objet),
- la vue de profil droit : (ProfilDroit(Objet)),
- la vue de face : VueFace(Objet),
- la vue de dos d'étiquette : VueDos(Objet),
- la vue de trois quart gauche de dos : TroisQuartGaucheDos(Objet),
- la vue de trois quart droit de dos : TroisQuartDroitDos(Objet),
- la vue de trois quart gauche de face : TroisQuartGaucheFace(Objet),
- la vue de trois quart droit de face : TroisQuartDroitFace(Objet),

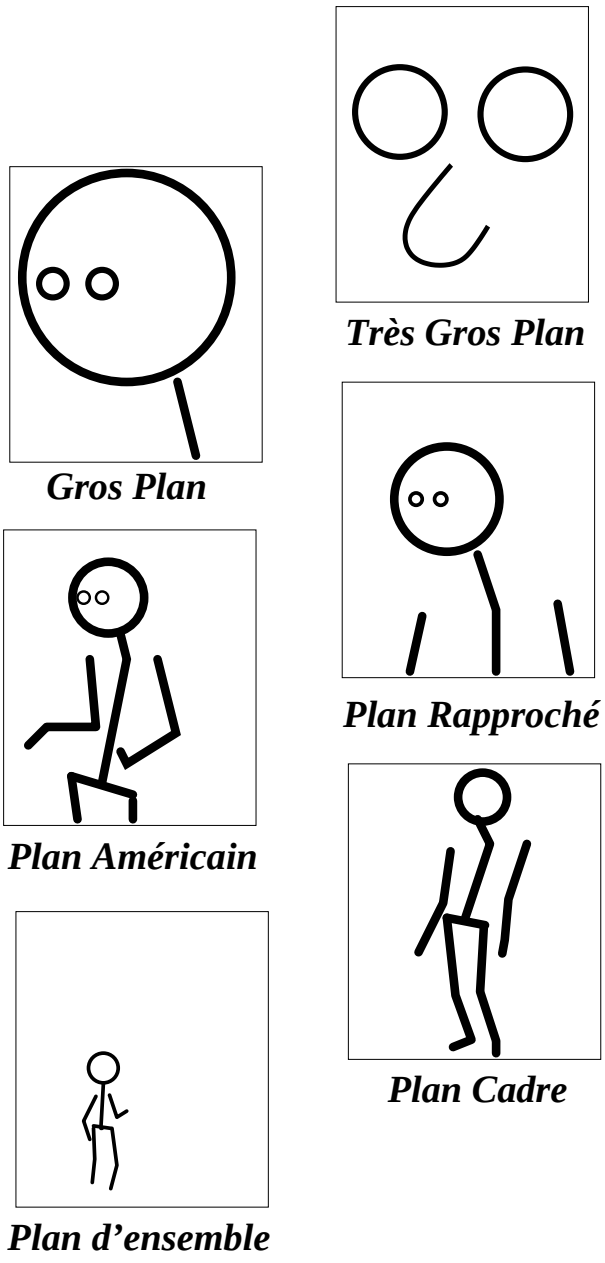



Figure 4.5 – Les six distances usuelles utilisées pour le cadrage en cinématographie, d'après [Ari76, Chr03, ww5w].

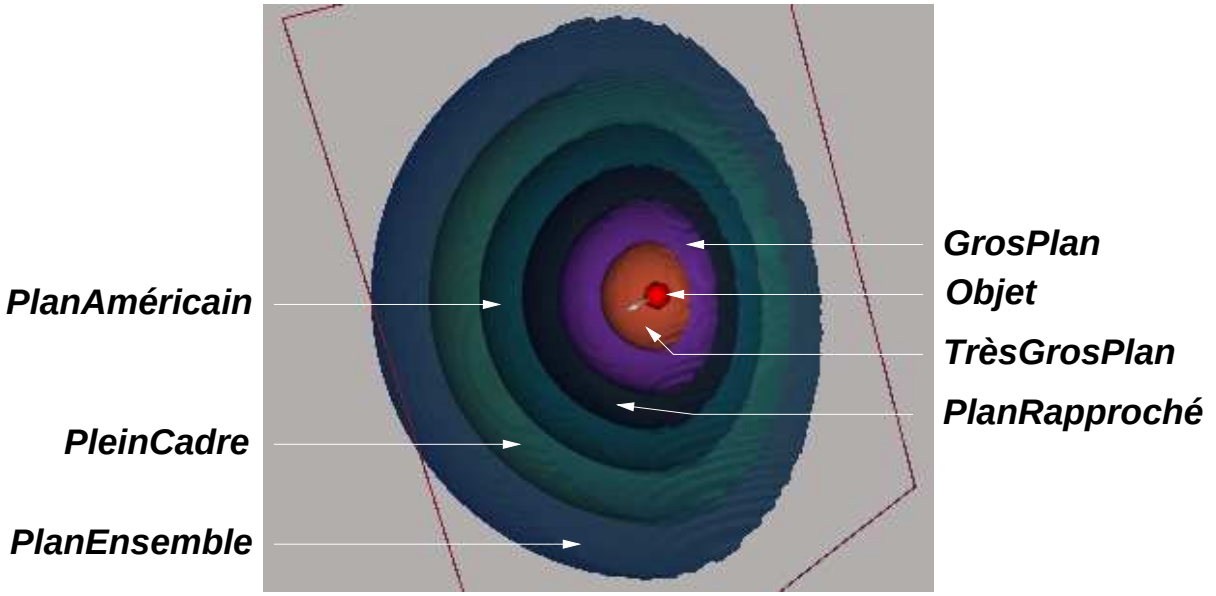



Figure 4.6 – Volumes sémantiques associés à la notion d'échelle de plans de vue pour un objet et induits par la propriété de projection.

- la vue en plongée : Plongée(Objet),
- la vue en contre-plongée : ContrePlongée(Objet).

Les positions de caméra correspondantes sont représentées en figure 4.7 et sont calculées relativement à un objet ayant un vecteur d'orientation intrinsèque (cf. Annexe A.0.2 sur les modèles de représentation des objets).

L'opérateur $G_f$ de filtrage géométrique appliqué à une propriété d'orientation construit les volumes sémantiques identifiés par la figure 4.7. Les frontières de ces volumes sont obtenues en calculant le produit scalaire entre le vecteur de l'objet considéré (la flèche sur le schéma 4.7) et un vecteur reliant le centre de l'objet à la position potentielle de caméra. Ce produit scalaire permet de déterminer l'angle de vue (dos, face, droite, gauche, etc.) sous lequel l'objet est perçu depuis la caméra. Une fois encore, nous autorisons une variation par rapport à l'orientation idéale afin d'ajouter de la flexibilité à notre méthode. La figure 4.8 illustre la construction d'un sous-ensemble des volumes en 3D caractérisant une propriété d'orientation, à savoir ceux correspondant aux orientations de face, de dos et de profils droit et gauche.

Bien que les seules orientations concernant le profil d'un obejt proposées nativement par la méthode des volumes sémantiques sont celles de profil droit et profil gauche, l'expressivité de notre méthode permet de caractériser un volume correspondant à la vue du profil d'un objet. Cette propriété est alors caractérisée par le volume sémantique $v_{s_{profil}} = \langle \mathcal{S}_{profil}, \mathcal{V}_{profil} \rangle$ définit en fonction des deux volumes sémantiques représentant les profils droit et gauche de cet objet. En effet, pour obtenir l'orientation de profil d'un objet, il suffit de créer les composantes du volume sémantique comme suit :

$$\mathcal{S}_{profil} = \mathcal{S}_{profilDroit} \wedge \mathcal{S}_{profilGauche}$$

et

$$\mathcal{V}_{profil} = \mathcal{V}_{profilDroit} \cup \mathcal{V}_{profilGauche}$$

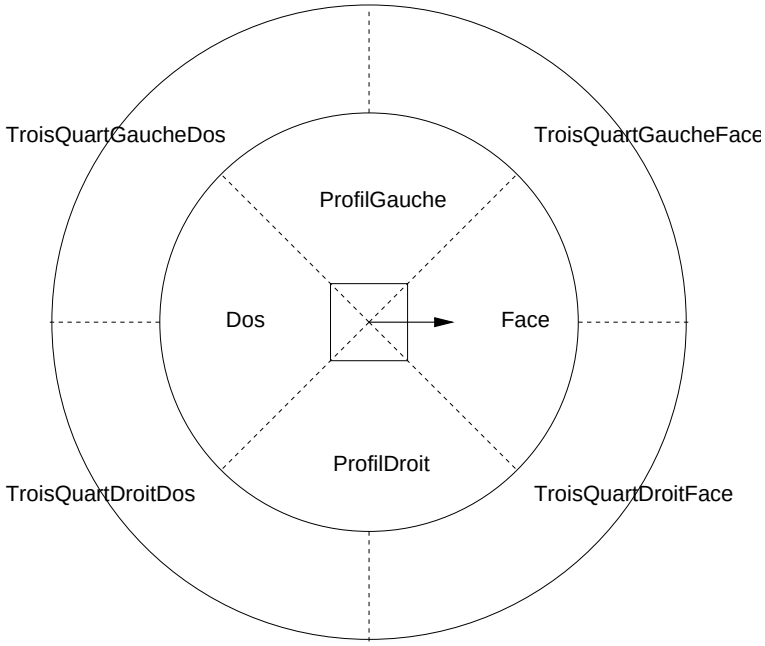



Figure 4.7 – Les principaux angles de vue utilisés en cinématographie, d'après [Ari76, Chr03].

L'étiquette sémantique associée au volume sémantique « profil » correspond à la conjonction des étiquettes sémantiques ProfilDroit(Objet) et ProfilGauche(Objet). Le volume des positions de caméra correspondantes représente quant à lui l'union des volumes représentant les vues de profil droit et gauche.

### 4.2.3 La propriété d'occlusion

Cette propriété permet de spécifier les relations spatiales de recouvrement (ou occlusions) d'objets à l'écran. Une occlusion entre deux objets $A$ et $B$ correspond à la superposition des projections des deux objets d'intérêt dans l'image résultat. Ainsi, nous disons que $A$ occulte $B$ (respectivement $B$ occulte $A$) si la projection de l'objet $A$ (resp. $B$) recouvre celle de l'objet $B$ (resp. $A$) à l'écran, c'est-à-dire si l'objet $A$ (resp. $B$) se trouve plus proche de la caméra que $B$ (resp. $A$).

La propriété d'occlusion offre à l'utilisateur la possibilité de spécifier des contraintes de visibilité entre un couple d'objets à l'intérieur d'une scène 3D. Les variations dans les degrés d'occlusion illustrés en figure 4.9 nous amènent à proposer trois types de propriétés à l'utilisateur :

- occlusion totale : un objet cache complètement la projection d'un autre objet à l'écran,
- occlusion partielle : la projection d'un objet recouvre en partie celle de l'autre objet mis en jeu dans la propriété,
- absence d'occlusion : l'utilisateur souhaite spécifier qu'un objet ne soit pas occulté par un autre à l'écran.

En appliquant cette propriété à tous les couples d'objets de la scène et en spécifiant à chaque fois le type d'occlusion souhaité (parmi les trois proposés plus haut), le réalisateur virtuel peut définir toutes les combinaisons d'occlusions totales, partielles ou de non occlusion dans la scène.

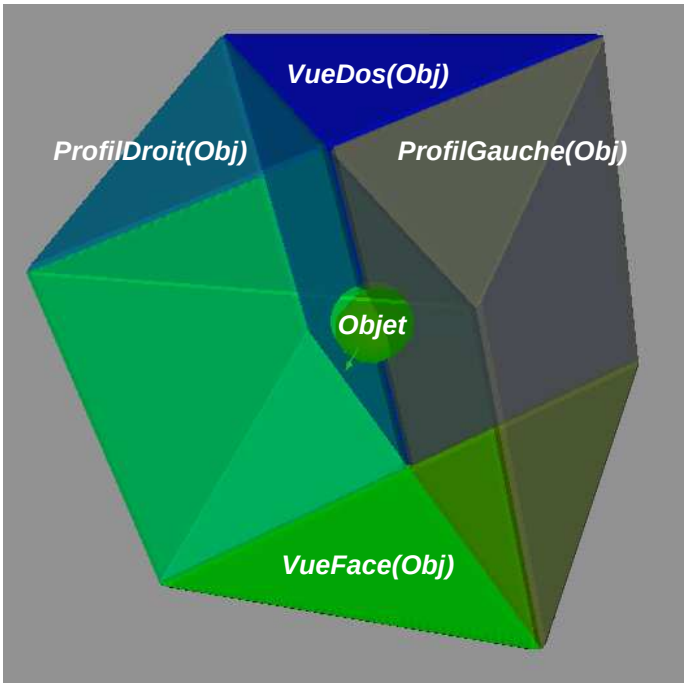



Figure 4.8 – Propriétés d'orientation définies pour quatre angles de vue communs et illustration des étiquettes sémantiques associées.

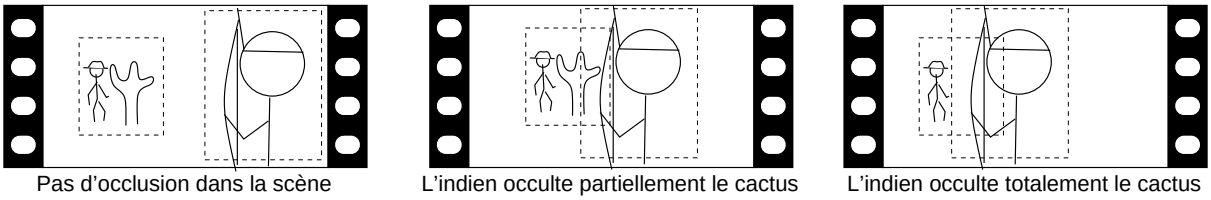



Figure 4.9 – Différents degrés d'occlusion dans une scène.

Le phénomène d'occlusion n'est pas détaillé plus précisément ici, le chapitre 5 y étant consacré. Toutefois, nous pouvons remarquer que l'occlusion est une relation purement géométrique. Une occlusion apparaît dès lors qu'une partie de la projection d'un objet à l'écran est recouverte par celle d'un autre objet d'intérêt. En instanciant cette remarque à un écran (*i.e.* une grille de pixels), nous pouvons discrétiser le phénomène d'occlusion, en remarquant qu'il y a :

- occlusion partielle : dès lors qu'un pixel de la projection d'un objet est recouvert par la projection de l'autre objet impliqué dans la propriété,
- occlusion totale : lorsque tous les pixels de la projection d'un objet sont recouverts par ceux de l'autre objet d'intérêt,
- absence d'occlusion : lorsque les projections des deux objets sont disjointes à l'écran.

Notons que nous ne prenons ici en compte que les aspects purement géométriques de l'occlusion, c'est-à-dire de recouvrement de projections d'objets. Toutefois, nous pouvons également considérer l'aspect sémantique d'identification d'objets lié à l'occlusion. Ceci nous incite à considérer l'occlusion en termes de reconnaissance d'objets à l'image et donc déclarer un objet comme n'étant pas occulté si un observateur est capable de le reconnaître sans ambiguïtés à l'écran.

Les volumes sémantiques induits par application de l'opérateur $G_f$ à une propriété d'occlusion sont calculés à partir de cônes caractéristiques définis selon les positions 3D des deux objets impliqués dans la propriété d'occlusion. La création de ces cônes s'inspire des travaux de Durand *et al.* [DDP02] sur le complexe de visibilité 3D et est présentée en détails dans la section 4.5.3.3. Les volumes intérieurs des cônes définissent les volumes d'occlusion (partielle ou totale), tandis que les volumes à l'extérieur des cônes correspondent aux zones de non occlusion entre objets, comme le montre la figure 4.10.

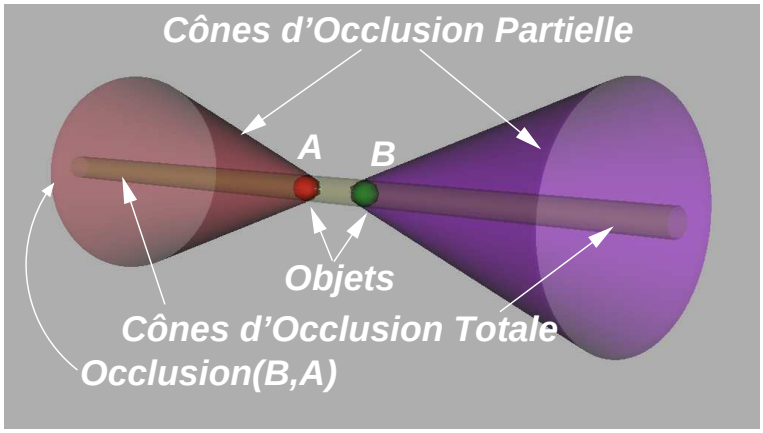



Figure 4.10 – Volumes sémantiques générés pour une propriété d'occlusion définie sur deux obejts d'intérêt $A$ et $B$.

### 4.2.4 Propriété de positionnement relatif à l'écran

La propriété de positionnement relatif d'objets à l'écran consiste à définir des relations spatiales à l'écran entre un couple d'objets $(A, B)$, *e.g.* « $A$ est à droite de $B$ », etc.

Le positionnement relatif d'objets va s'effectuer par rapport à des plans de coupe définis dans l'es-

pace de recherche, en fonction de la position des objets dans la scène. La propriété de positionnement gauche/droite d'un couple d'objets à l'écran est illustrée en figure 4.11. Il s'agit de positionner un plan $\mathcal{P}$ passant par les centres des deux objets impliqués dans la propriété, séparant en deux l'espace des positions possibles de la caméra. Nous voyons en effet que la caméra $\text{cam}_1$ est située dans une zone de l'espace $\mathsf{EstÀDroiteDe}(O_1, O_2)$, pour laquelle $O_1$ apparaît à droite de $O_2$ dans l'image résultat et ce quelque soit son orientation (sous réserve d'apparition des objets à l'écran). À l'inverse, la caméra $\text{cam}_2$ aboutit à une image dans laquelle $O_1$ est à gauche de $O_2$, elle appartient à la région $\mathsf{EstÀGaucheDe}(O_1, O_2)$.

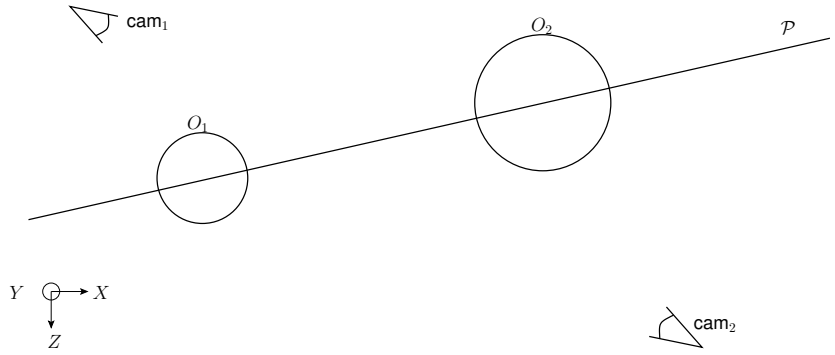



Figure 4.11 – Vue de dessus d'un plan de coupe définit pour la propriété de positionnement relatif « Gauche/Droite » des objets $O_1$ et $O_2$ à l'écran.

L'avantage de cette propriété est qu'elle ne nécessite pas de placement fin des objets à l'écran, elle est donc beaucoup moins contraignante que la propriété de cadrage qui consiste à définir un dadre dans l'image à l'intérieur duquel doit apparaître un objet. L'utilisateur peut envisager d'avoir recours à une propriété de positionnement relatif afin de spécifier l'agencement à l'écran d'objets de moindre importance, mais dont il souhaite tout de même pouvoir contrôler les relations spatiales.

### 4.2.5 Propriété de cadrage (ou *framing*)

La propriété de cadrage (également appelée propriété de *framing*) permet au metteur en scène virtuel de spécifier les opérations de cadrage issues de la cinématographie classique. Celles-ci permettent de déterminer les objets qui apparaissent à l'image et ceux que le réalisateur désire laisser hors-cadre (ou hors-champ), c'est-à-dire qui n'appartiennent pas à l'image finale. Cette propriété permet de contraindre un objet dans un cadre (ou *frame*) spécifié dans l'espace écran (cf. figure 4.12). Celui-ci peut être défini à l'intérieur, partiellement à l'intérieur ou en dehors de l'espace image.

L'expressivité de cette propriété permet de définir à la fois les positions et les tailles relatives des objets à l'écran. Une propriété de *framing* contraint ainsi à la fois la position et l'orientation de la caméra et peut être, de ce fait, source d'inconsistances potentielles dans la description d'un problème de placement de caméra.

Nous définissons la propriété de cadrage à partir des coordonnées du cadre devant contenir l'objet d'intérêt (plus particulièrement les coordonnées, dans l'espace écran, du coin en bas à gauche, et en haut à droite de la *frame*), et de l'objet en question.

De plus, les propriétés de *framing* peuvent également impliquer des propriétés d'occlusion en fonc-

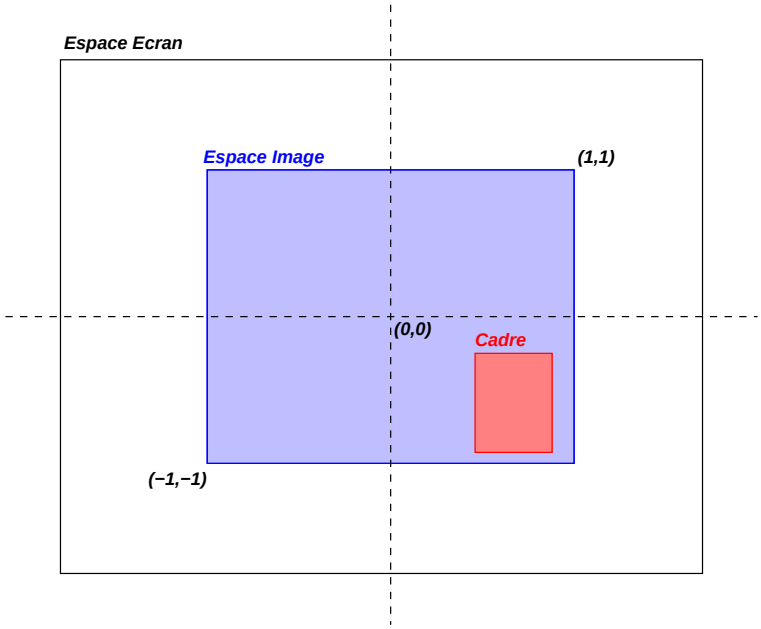



Figure 4.12 – L'espace écran et l'espace image.

tion de leur placement dans l'espace image. En effet, si l'utilisateur définit des cadres se chevauchant dans l'espace image (cf. figure 4.9), alors nous pouvons inférer que ce dernier désire une occlusion (partielle ou totale) entre les objets mis en jeu dans ces propriétés. *A contrario*, une absence de recouvrement entre plusieurs *frames* indique une absence d'occlusion entre les objets en présence.

La construction des volumes sémantiques associés à une propriété de *framing* dépend du nombre de cadres définis par l'utilisateur pour une même image, ainsi que de leur positions relatives à l'intérieur de l'espace image. Cette construction nécessite une étude de cas :

- une seule propriété de cadrage a été définie : nous la considérons dans ce cas comme une propriété de projection. La taille du cadre ainsi que celle de l'objet (ou de son englobant) dans la scène 3D nous fournissent les paramètres nécessaires à la définition d'une propriété de projection, limitant ainsi la distance entre la caméra et l'objet dans la scène. Notons que pour une propriété de cadrage, toutes les orientations de l'objet d'intérêt sont permises.
- au moins deux propriétés de *framing* ont été définies par l'utilisateur : une étude de cas doit alors être menée pour chaque couple de propriétés afin d'extraire les informations supplémentaires contenues dans la description de l'utilisateur.

Plutôt que de réaliser une étude exhaustive de tous les cas de figure pouvant se présenter, nous préférons nous limiter ici à la présentation de deux configurations possibles qui permettent au lecteur d'inférer la gestion des cas restants. Tout d'abord, nous nous intéressons à une description de cadres sans recouvrements, puis nous nous attachons à détailler la construction des volumes sémantiques correspondants à un chevauchement de cadres.

Rappelons que les volumes sémantiques issus du processus de filtrage géométrique décrit ci-après contiennent les positions de caméra satisfaisant les relations spatiales entre les différents cadres impliqués dans les propriétés. Les positions des objets à l'intérieur des *frames* sont déterminées lors du processus

numérique, présenté en section 4.5.4 déterminant l'orientation des caméras résultats.

Ainsi, en se basant sur le problème décrit en figure 4.14, l'objet $A$ inclus dans le cadre 1 apparaît à gauche de l'objet $B$ (intérieur au cadre 2) à l'écran. Cependant les positions exactes de $A$ et $B$ à l'intérieur des cadres 1 et 2 sont calculées uniquement lors du processus numérique.

#### *4.2.5.1 Configuration sans chevauchements de cadre*

L'exemple développé ici s'intéresse à une configuration de propriétés de cadrage sans chevauchement, comme présenté en figure 4.13.

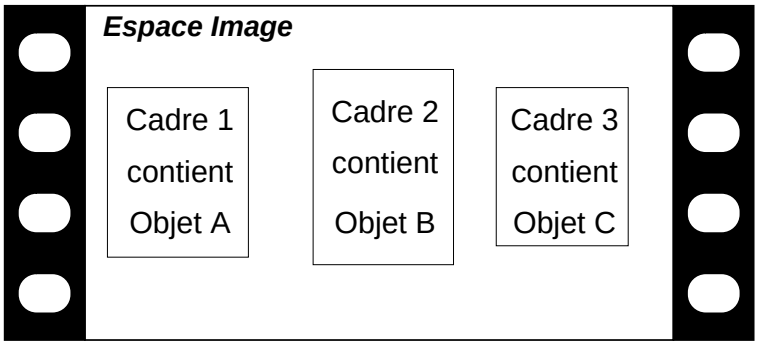



Figure 4.13 – Une description utilisateur impliquant trois cadres sans chevauchements.

Afin d'illustrer les méthodes mises en œuvre pour créer les différents volumes sémantiques impliqués dans cette description, nous nous basons sur la figure 4.14 qui présente une vue de dessus d'une scène 3D comportant trois objets $A$, $B$ et $C$. Pour des raisons de clarté, le schéma représente une en vue 2D de dessus de la scène d'intérêt. Toutefois, les calculs et les opérations décrites ci-après sont effectués en 3D.

Le premier volume sémantique est identifié par la Zone 1 (représentée en jaune dans la figure 4.14), située derrière les objets $A$ et $B$ dans la scène 3D. Ce volume est caractérisé par l'étiquette sémantique EstÀDroiteDe(A,B), qui signifie que l'objet $A$ apparaît à droite de l'objet $B$ dans l'espace image si la caméra est placée dans ce volume. Cette zone peut être éliminée puisque la description utilisateur spécifie que l'objet $A$ doit se trouver à la gauche de l'objet $B$ à l'écran.

La seconde information à retirer de la description est que les cadres 1 et 2 sont disjoints. L'image résultat ne doit donc comporter aucune occlusion entre les objets appartenant à ces *frames* (*i.e.* il ne doit pas avoir y occlusion entre $A$ et $B$). L'étape suivante consiste à créer les cônes d'occlusion partielles (cf. sous-section 4.2.3) afin de pouvoir identifier les volumes correspondants aux zones de non-occlusion entre $A$ et $B$. Ces cônes aboutissent à la génération des Zones 2 et 3 (respectivement de couleur verte et bleue sur le schéma), lesquelles sont respectivement étiquetées Occlusion(A,B) lorsque l'objet $B$ occulte $A$ et Occlusion(B,A) dans le cas contraire ($A$ occulte $B$).

Suite à l'analyse de la description utilisateur (limitée ici aux deux premiers cadres et donc aux objets $A$ et $B$), nous pouvons éliminer de l'espace de recherche les zones 1,2 et 3 car elles ne satisfont pas la description utilisateur. La zone de l'espace restante (la zone blanche située au bas de la figure) est donc considérée comme solution puisqu'elle répond pleinement à la description faite par l'utilisateur et est étiquetée (EstÀGaucheDe(A,B)). En effet, si nous plaçons la caméra dans cet espace solution (et sous

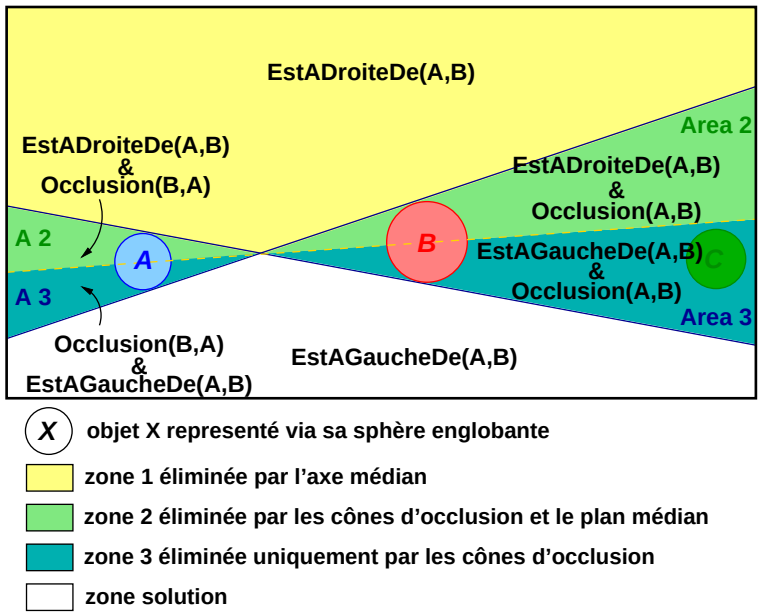



Figure 4.14 – Vue de dessus (2D) des volumes sémantiques concernant le couple d'objets ($A$,$B$) respectivement cadrés à gauche et à droite dans l'image résultat.

réserve que son orientation soit correctement calculée), l'objet $A$ apparaît à gauche de $B$ dans l'image résultat.

Ce processus doit être répété pour chaque couple d'objets de la scène impliqué dans une propriété de *framing*. Ici nous devons reproduire les mêmes raisonnements pour les couples d'objets $(A, C)$ et $(B, C)$. Le volume sémantique solution de la description illustrée en figure 4.13 est obtenu en calculant l'intersection des trois volumes respectivement étiquetés EstÀGaucheDe(A,B), EstÀGaucheDe(A,C) et EstÀGaucheDe(B,C). Ce volume solution correspond ainsi à la conjonction de ces trois étiquettes sémantiques, soit :

$$\text{EstÀGaucheDe(A,B)} \wedge \text{EstÀGaucheDe(A,C)} \wedge \text{EstÀGaucheDe(B,C)}$$

Pour proposer une solution à l'utilisateur, il suffit d'appliquer le processus numérique (cf. section 4.3.3) à l'intérieur de ce volume solution, afin de calculer une orientation correcte de la caméra.

#### *4.2.5.2 Configuration avec chevauchements de cadres*

La figure 4.15 illustre un exemple de description utilisateur comprenant trois propriétés de *framing* dont deux se chevauchent, correspondant au deuxième cas de figure présenté ci-avant.

La zone de recouvrement des cadres 1 et 2 est partielle. En effet, nous avons à la fois Cadre 1 $\subsetneq$ Cadre 2 et Cadre 2 $\subsetneq$ Cadre 1 dans l'espace écran. Nous pouvons ainsi inférer que l'utilisateur souhaite spécifer une occlusion partielle entre les objets $A$ et $B$. Nous ne présentons pas dans cette thèse d'exemples impliquant une occlusion totale entre deux objets, mais un utilisateur peut tout à fait spécifier une telle propriété. Pour ce faire, il lui suffit de déclarer deux cadres ($F_1$ et $F_2$) contenant chacun un objet ($O_1$

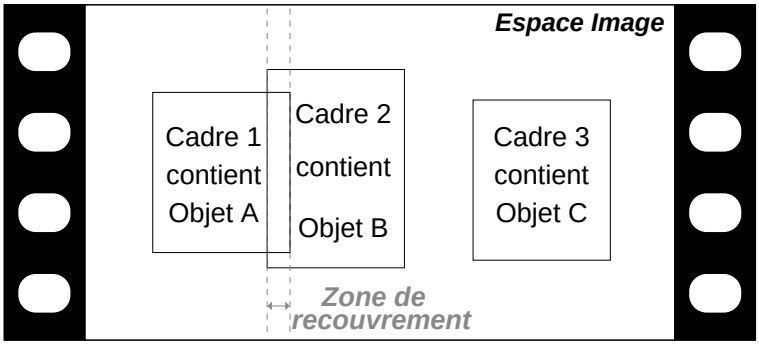



Figure 4.15 – Une description utilisateur comportant des propriétés de *framing* avec chevauchements des cadres.

et$O_2$), puis de superposer les *frames* afin que l'une d'elle recouvre totalement l'autre ($F_1 \subseteq F_2$ ou $F_2 \subseteq F_1$).

Il semble évident que l'expressivité liée à nos propriétés est limitée. En effet, si nous faisons référence aux descriptions précédentes (d'une occlusion totale ou partielle par superpositions de *frames*), l'utilisateur ne peut spécifier quel objet doit être devant l'autre, les deux cadres ne comportant aucune information de profondeur. Une solution envisagée pour résoudre ce problème est d'inclure dans le mécanisme de description la notion de calques pour les propriétés. Cela consiste à adjoindre à chaque propriété un chiffre représentant son niveau de profondeur dans la scène. Donner à chaque propriété le même numéro reviend au système existant, tandis que spécifier un numéro différent permet de différencier leur ordre d'apparition à l'écran. Prenons l'exemple de deux propriétés de cadrage, $P_{C_1}$ et $P_{C_2}$, contraignant respectivement les objets $O_1$ et $O_2$ dans deux cadres $F_1$ et $F_2$. Soit la propriété $P_{C_1}$ de profondeur 1, supposons que $P_{C_2}$ soit de niveau 2 et que $F_1 \subseteq F_2$. Cette description utilisateur correspond à une occlusion totale de $O_2$ par $O_2$.

L'implémentation actuelle de la méthode des volumes sémantiques ne permet pas de spécifier des relations de profondeur entre les différentes propriétés. Ainsi dans l'exemple précédent, nous sommes incapables de savoir si l'utilisateur souhaite voir $O_1$ occulté par $O_2$ ou si il spécifie l'occlusion inverse. Toutefois, l'expressivité proposée par la méthode des volumes sémantiques nous permet de fournir à l'utilisateur les deux classes de solutions, correspondant respectivement aux cas où $O_1$ occulte $O_2$ et où $O_2$ occulte $O_1$, comme nous le présentons maintenant.

Comme précédement, nous basons notre explication sur un schéma présentant une vue de dessus d'une scène 3D correspondant à la description illustrée en figure 4.15.

La section précédente a illustré le fait qu'une description utilisateur dans laquelle des cadres se chevauchent induit une occlusion partielle entre les objets impliqués. Nous devons donc interdire les zones de l'espace de recherche pour lesquelles ces deux objets ont des projections disjointes, tout comme celles correspondantes aux régions à l'intérieur desquelles un objet occulte complètement un autre (la projection de $A$ recouvre totalement celle de $B$ ou inversement). Pour ce faire, il faut tout d'abord construire les cônes d'occlusion partielle et totale, comme présenté précédemment. Les zones 1,2,3,4 et 5 correspondent aux régions pour lesquelles il y a occlusion totale soit de $A$ par $B$, soit de $B$ par $A$. Nous leur associons les étiquettes sémantiques OcclusionTotale(A,B) et OcclusionTotale(B,A).

La zone centrale située entre $A$ et $B$ (et étiquetée Entre(A,B)) est inconsistante puisqu'elle se situe

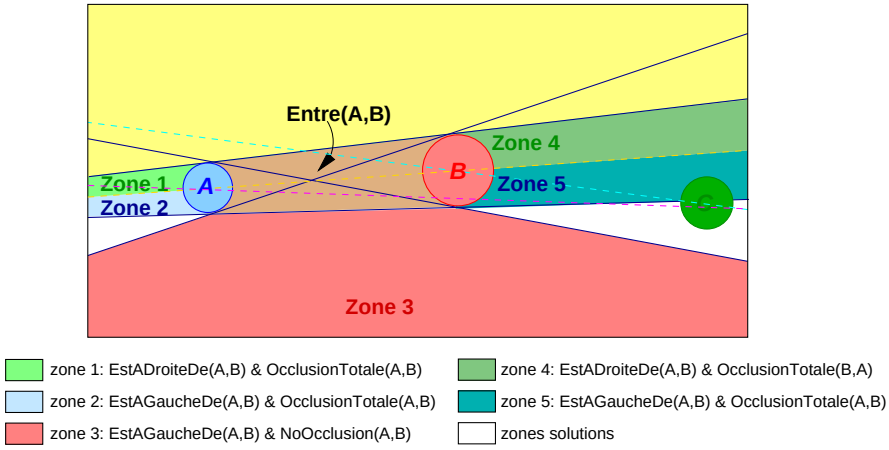



Figure 4.16 – Vue de dessus (2D) des volumes sémantiques concernant le couple d'objets $(A,B)$ obtenus d'après la description illustrée en figure 4.15.

à l'intérieur des cônes d'occlusion totale. De plus, positionner une caméra à l'intérieur de ce volume ne permet pas de voir à l'écran à la fois les objets $A$ et $B$ en utilisant une distance focale ne déformant pas énormément l'image résultat.

La zone 3 de la figure 4.16 est également interdite puisque elle correspond à des positions de caméra n'amenant à aucune occlusion partielle à l'écran. Bien évidemment, ici encore, nous prenons en compte le fait que le Cadre 1 se situe à gauche du Cadre 2 dans la description utilisateur. De ce fait, les zones éliminées dans la sous-section précédente le sont également dans cet exemple. Par souci de clarté nous ne re-détaillons pas ici pourquoi elles ont été évincées.

Si nous dressons un bilan des régions éliminées, nous observons que toutes les zones vertes et bleues de la figure 4.16 ainsi que les zones rouges, rosées et jaunes l'ont été pour les raisons évoquées plus haut. L'unique région restante (représentée en blanc sur la figure 4.16) correspond à une zone solution de la description utilisateur. Toutefois, pour respecter complètement la description de l'utilisateur, il nous faut prendre en compte les couples d'objets $(A,C)$ et $(B,C)$. Les lignes pointillés violette et bleue turquoise illustrent les plans permettant de décider du placement à l'écran des couples d'objets $(A,C)$ et $(B,C)$ respectivement. Dans ce cas, l'ajout des contraintes de placement relatif entre les objets élimine encore une partie de la zone blanche située à droite, sous l'objet $C$. Nous obtenons alors deux régions non-connexes de l'espace de recherche initial, répondant toutes deux à la description utilisateur.

Ces deux zones illustrent parfaitement l'originalité et l'apport des volumes sémantiques comparé aux approches existantes de placement de caméra en environnements vituels. En effet, la description de l'utilisateur étant approximative et/ou imprécise (dans le cas présent du fait du manque d'expressivité du format de description de nos propriétés), il existe plusieurs solutions sémantiquement équivalentes au problème formulé. Les niveaux de profondeur entre propriétés n'étant pas implémentés dans la version actuelle des volumes sémantiques, nous ne savons pas quel objet doit occulter l'autre dans la description utilisateur. Ainsi, les solutions pour lesquelles $A$ occulte partiellement $B$ sont sémantiquement équivalentes à celles où $B$ occulte $A$. Les approches existantes de placement de caméra ne proposent qu'une seule solution aux utilisateurs, éclipsant ainsi une des deux classes de solutions du problème. Au contraire, les volumes sémantiques permettent de présenter deux solutions caractéristiques de chacune

des classes de solutions. Nous pouvons ainsi proposer à l'utilisateur une solution pour laquelle $A$ apparaît devant $B$ (occlusion partielle de $B$ par $A$) et une autre où $B$ est plus proche de la caméra.

## 4.3 Processus de résolution

Une fois présentées toutes les propriétés offertes à l'utilisateur, nous pouvons détailler le processus de résolution utilisé lors de la construction des volumes sémantiques. Comme nous l'avons présenté dans l'introduction (section 4.1), le processus de résolution est composé de quatre étapes :

1. la description du problème par l'utilisateur *via* des propriétés cinématographiques sur les objets de la scène 3D,
2. la création et l'intersection des volumes sémantiques répondant au problème,
3. le calcul des meilleurs candidats pour chacun des volumes sémantiques solutions,
4. la présentation des résultats à l'utilisateur.

Nous allons maintenant décrire chacune de ces quatre étapes.

### 4.3.1 Description du problème

Cette phase correspond à la spécification du résultat souhaité par l'utilisateur. Pour ce faire, celui-ci doit spécifier les relations entre objets grâce aux propriétés décrites dans la section précédente. Dans notre méthode des volumes sémantiques, nous devons connaître toutes les informations relatives à la scène 3D étudiée, c'est-à-dire les positions, les vecteurs d'orientation intrinsèques et les tailles des volumes englobants (cf. section A.0.2 sur la représentation des objets) de chacun des objets d'intérêt.

Dans l'implémentation actuelle de l'approche des volumes sémantiques (cf. section 4.5), la description des propriétés se fait par le biais d'un langage de script. L'utilisateur spécifie une liste de propriétés cinématographiques, en fonction des objets présents dans la scène.

Nous avons également envisagé de proposer une interface graphique permettant la description d'un problème de composition visuelle, en particulier en proposant des greffons (*plug-ins*) pour les principaux modeleurs du marché (MAYA, 3D STUDIO MAX, etc.). Toutefois, cette interface graphique n'est pas implémentée dans le système actuel, nous y revenons dans la section 4.8 consacrée aux perspectives.

### 4.3.2 Le partitionnement sémantique de l'espace : le processus géométrique

L'application du partitionnement sémantique fournit un ensemble (potentiellement vide) de volumes sémantiques : $\{v_{s_0}, \ldots, v_{s_n}\}$, avec $\forall i, v_{s_i} = \langle \mathcal{S}_i,\ \mathcal{V}_i \rangle$ (cf. définition 4.1). Du fait des propriétés de correction de l'opérateur $G_f$ et du processus d'intersection, l'absence de volume sémantique à l'issue de l'étape de partitionnement sémantique nous permet de conclure à l'inconsistance du problème. Toutefois, en aucun cas l'existence d'un volume n'atteste de la présence de solutions, puisque l'orientation de la caméra doit encore être déterminée.

Comme détaillé dans la section précédente, un problème de placement de caméra est modélisé par un ensemble de propriétés cinématographiques $p_i$. La résolution d'un problème de placement de caméra

consiste au traitement de la conjonction des $p_i$, c'est-à-dire :

$$\begin{aligned} \bigcup_i p_i &= \bigcap_i G_f(p_i) \\ &= \bigcap_i \langle \mathcal{S}_i, \mathcal{V}_i \rangle \\ &= \langle \bigwedge_i \mathcal{S}_i, \bigcap_i \mathcal{V}_i \rangle \end{aligned}$$

Ainsi, le(s) volume(s) solution(s) d'une description utilisateur correspond(ent) à l'intersection Booléenne des volumes sémantiques ($\mathcal{V}_i$) de chacune des propriétés. De même, l'étiquette sémantique associée correspond à la conjonction des étiquettes $\mathcal{S}_i$.

Le partitionnement sémantique de l'espace de recherche, aboutissant à la construction des volumes sémantiques $v_{s_i}$ (associés aux propriétés $p_i$) permet à la fois d'identifier une inconsistance à partir de la description utilisateur et dans le cas contraire, de déterminer le nombre de classes de solutions du problème de placement de caméra. En effet, deux cas de figure se présentent :

- l'existence de plusieurs volumes sémantiques disjoints : il existe différentes classes de solutions pour le problème spécifié. Chacune d'entre elles possède des caractéristiques distinctes par rapport aux objets composant la scène 3D. Ces dernières sont identifiées en étudiant les étiquettes sémantiques associées à chaque volume non-connexe. Nous devons appliquer un processus numérique à chacun de ces volumes, afin de calculer le meilleur représentant de chaque classe de solutions.
- l'absence de tout volume sémantique : le problème est sur-contraint. Ce cas de figure illustre l'avantage de la méthode des volumes sémantiques par rapport aux approches existantes. En effet, sans avoir à mettre en œuvre aucun processus numérique nous pouvons conclure à l'inconsistance du problème énoncé.

### 4.3.3 Calcul de configurations consistantes : le processus numérique

À partir d'un ensemble de volumes sémantiques, le processus numérique détermine le meilleur représentant de chacun d'eux. Cette étape consiste à calculer, à l'intérieur de chacun des $\mathcal{V}_i$, une configuration de caméra consistante, tant au niveau de la position, obtenue par construction des volumes, que de l'orientation et de la distance focale. Cette configuration doit à la fois maximiser la satisfaction des propriétés utilisateur et satisfaire :

- les propriétés de cadrage,
- les positionnements relatifs,
- les orientations.

La satisfaction des propriétés est exprimée par le biais d'une fonction objectif associée à chaque propriété. Le processus d'optimisation est guidé par une fonction de coût définie comme une agrégation de la satisfaction de chacune des propriétés, c'est-à-dire de chacune des fonctions objectif.

**Exemple 4.4 (Maximisation de la satisfaction d'une propriété cinématographique).** *Soit la propriété cinématographique d'orientation suivante :*

*« Voir Calvin de profil gauche ».*

*Nous associons à cette dernière l'étiquette sémantique :* « ProfilGauche(Calvin) ».
*Les meilleurs représentants possibles (dont un exemple est représenté en figure 4.17) correspondent aux configurations positionnées sur une droite perpendiculaire au vecteur orientation de Calvin (droite en pointillés) et orientées de façon à regarder son profil gauche (zone délimitée par les droites). Ceci peut être trivialement exprimé par un produit scalaire exprimé entre l'orientation de Calvin (représentée par un flèche sur la figure) et un vecteur reliant Calvin à la position de la caméra.*

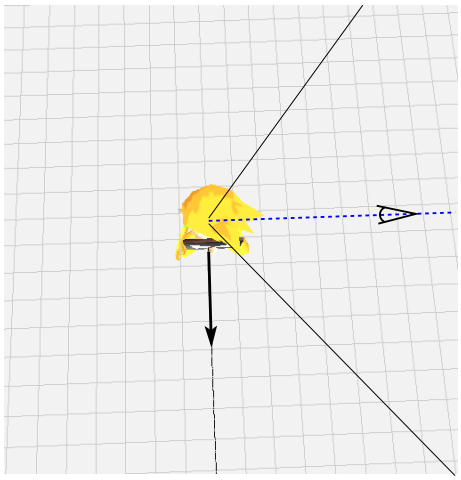


Figure 4.17 – Illustration d'une position de caméra maximisant la propriété « Voir Calvin de profil gauche ».

Maximiser la satisfaction d'une propriété revient donc à minimiser les fonctions de coût associées à celle-ci. Le problème du calcul de configurations consistantes consiste à déterminer un septuplet $c$ de valeurs (correspondant aux sept degrés de liberté de la caméra, cf. annexe A.0.1), tel que :

$$\begin{cases} \min \sum_i \mathsf{cost}_{p_i}(c) \\ \forall j, f_j(c) \text{ est satisfaite} \\ c \in \mathcal{V}_i \end{cases}$$

Où $\mathsf{cost}_{p_i}(c)$ correspond à la fonction de coût associée à la propriété $p_i$ et $f_j$ représente la $j^e$ propriété de *framing* spécifiée par l'utilisateur.

Comme nous l'avons présenté dans le chapitre précédent, les approches classiques d'optimisation ou d'optimisation sous contraintes, requièrent le plus souvent des fonctions objectif différentiables (*e.g.* descente sous gradients). De telles méthodes ne sont donc pas applicables pour la méthode des volumes sémantiques puisque certaines propriétés ne sont pas exprimables algébriquement (non appartenance à une frame par exemple). De plus, nous devons prendre en compte la contrainte non algébrique d'appartenance d'un point à un volume 3D ($c \in \mathcal{V}_i$). Nous proposons ainsi d'utiliser l'algorithme de Recherche Locale Continue (RLC) numérique présenté dans le chapitre précédent, en section 3.3.1.5.

## 4.4 Exploitation des volumes sémantiques

La contribution principale de l'approche des volumes sémantiques est d'offrir une nouvelle base d'exploration et d'interaction avec les solutions d'un problème de composition visuelle. Nous devons exhiber chacune des classes de solutions existantes, afin que l'utilisateur puisse se rendre compte des placements de caméra possibles satisfaisant les propriétés cinématographiques qu'il a spécifié. Nous illustrons par la suite trois types différents d'interactions possibles :

1. augmenter les informations disponibles sur un volume sémantique donné,

2. caractériser un volume sémantique par rapport à un objet donné,
3. raisonner sur les volumes sémantiques.

### 4.4.1 Caractérisation d'un volume sémantique

Pour une description donnée, chaque volume sémantique fourni par le solveur géométrique (l'opérateur $G_f$) intègre la connaissance relative à la satisfaction des propriétés. À partir de cette connaissance, la caractérisation de chacun des volumes peut être augmentée en fonction de propriétés que l'utilisateur n'a pas spécifiées. Prenons par exemple un volume sémantique $v_s$ caractérisé par la propriété « Voir l'objet $A$ de profil gauche ». Nous pouvons caractériser plus finement ce volume en considérant les positions « Face » et « Dos » de l'objet d'intérêt. Nous obtenons ainsi à partir de $v_s$ deux sous-volumes sémantiques $v_{s_1}$ et $v_{s_2}$ qui correspondent aux propriétés « Voir l'objet $A$ de profil gauche et de face » et « Voir l'objet $A$ de profil gauche et de dos ».

La description du problème n'a pas changée, mais les volumes $v_{s_1}$ et $v_{s_2}$ sont « sémantiquement augmentés » par rapport à $v_s$. Nous pouvons présenter à l'utilisateur les différences existantes entre ces deux sous-volumes, ce qui reviend à établir la liste des étiquettes sémantiques de $v_{s_1}$ et $v_{s_2}$ non satisfaites par le volume $v_s$, c'est-à-dire $v_{s_1} \cap v_{s_2}$.

### 4.4.2 Caractérisation de la scène 3D

Notre méthode de placement de caméra en environnements virtuels permet d'obtenir une caractérisation complète de la scène 3D en spécifiant pour chaque objet ou couple d'objets, des propriétés cinématographiques aboutissant à la construction d'un ensemble de volumes sémantiques. Cette possibilité permet à l'utilisateur d'effectuer des requêtes concernant des propriétés définies sur un objet ou un couple d'objets qu'il n'avait pas inclus dans la description initiale du probème de placement de caméra. Chacune de ces requêtes aboutit à l'application du processus de filtrage géométrique, c'est-à-dire la création des volumes par l'opérateur $G_f$ de filtrage géométrique et leur intersection Booléeenne. Le nombre de composantes connexes de cette intersection indique le nombre de classes de solutions issues de cette caractérisation des objets. Comme précédement, une intersection vide induit une inconsistance dans la description.

**Exemple 4.5 (Caractérisation de volumes sémantiques).** *Soit un volume sémantique $v_s$ issu d'une description « Voir l'objet A de face ». L'utilisateur décide de raffiner le problème initial en fonction des objets $B$ et $C$, le problème de placement de caméra devient : « Voir l'objet $A$ de face ET Voir l'objet $B$ de face ET Voir l'objet $C$ de face ».*
*Cette nouvelle description nous permet de calculer le volume sémantique solution, en appliquant l'opérateur $G_f$ :*

$$v'_s = v_s \cap G_f(\mathsf{VueFace(B)}) \cap G_f(\mathsf{VueFace(C)})$$

*Si $v'_s$ est vide, la nouvelles description est inconsistante, il n'existe pas de configuration de caméra répondant au problème. Dans le cas contraire, $v'_s$ contiend les positions de caméra répondant possiblement au problème, le processus numérique doit y être appliqué afin de calculer une orientation cohérente et pouvoir présenter une configuration solution à l'utilisateur.*

Nous pouvons donc, à partir d'un problème, essayer d'augmenter la description utilisateur afin d'affiner les solutions plutôt que de spécifier le problème en une seule fois au risque de définir un problème sur-contraint. La figure 4.34 qui présente les résultats d'un de nos jeux de tests, illustre la prise en compte

de requêtes complémentaires, une fois les volumes sémantiques répondant à sa description initiale obtenus par le processus de partitionnement sémantique de l'espace.

### 4.4.3 Raisonnement sur les volumes sémantiques

Afin de fournir un mécanisme d'interaction de haut-niveau, d'augmenter le nombre de propriétés proposées par la méthode des volumes sémantiques et donc l'expressivité générale de notre approche, il est possible d'utiliser un système à base de règles permettant la réécriture des étiquettes sémantiques et calculant les volumes sémantiques associées à ces nouvelles descriptions.

**Exemple 4.6 (Raisonnement sur les volumes sémantiques).** *Soit la règle suivante :*

$$\frac{\mathsf{EstÀGaucheDe(A,B)} \wedge \mathsf{EstÀGaucheDe(B,C)}}{\mathsf{Entre(B,A,C)}}$$

*Celle-ci permet d'agréger deux propriétés de positionnement relatif d'objets à l'écran en une troisième n'appartenant pas à l'ensemble des propriétés gérées nativement par l'approche des volumes sémantiques.*
*On peut également effectuer l'opération inverse afin de séparer une propriété complexe en deux propriétés plus simples :*

$$\frac{\mathsf{Entre(B,A,C)}}{\mathsf{EstÀGaucheDe(A,B)} \wedge \mathsf{EstÀGaucheDe(B,C)}}$$

À partir d'un système à base de règles, il est possible d'étendre les étiquettes sémantiques basiques par de nouvelles propriétés obtenues par composition des propriétés cinématographiques de base et donc d'augmenter l'expressivité des volumes sémantiques.

## 4.5 Implémentation

Cette section est consacrée à l'implémentation de l'approche des volumes sémantiques. Nous avons choisi d'implémenter les solveurs géométriques et numériques en tant que librairies indépendantes. Le solveur géométrique calcule les volumes sémantiques associés aux propriétés, puis effectue les intersections adéquates. Le solveur numérique permet quant à lui, de calculer les meilleurs représentants d'un volume sémantique. Pour réaliser l'interface entre ces deux solveurs, nous avons eu recours au logiciel ZDM, un modeleur 3D développé par Marc Christie, brièvement présenté en annexe B.

Cependant, avant de nous attacher à la description de la mise en œuvre des propriétés et de l'algorithme de recherche des meilleures configurations à l'intérieur des volumes, nous nous intéressons au problème lié à l'intersection des volumes sémantiques.

Les opérations Booléennes entre modèles 3D, bien qu'implémentées dans tous les modeleurs modernes, sont des opérations délicates à mettre en œuvre pour des volumes en représentation polyédrique, de par la multitude de configurations pouvant amener à des cas pathologiques. La figure 4.18 illustre un exemple de singularité rencontrée lors de l'application d'une opération d'intersection ou de soustraction Booléenne sur deux modèles à facettes polyédriques.

Dans ce schéma, la face $f$ résultant d'une opération booléenne (soustraction ou intersection) des deux solides $s_1$ et $s_2$ est problématique étant coplanaire à deux faces de $s_1$ et $s_2$. La caractérisation de $f$ est difficile pour les modèles en représentation paramétrique. Pour de plus amples informations sur les

problèmes rencontrés en géométrie de construction de solides (*Constructive Solide Modelling — CSG*), le lecteur est invité à se référer au chapitre 3 du livre « Geometric and Solid Modeling: An Introduction » de Hoffmann [Hof89].

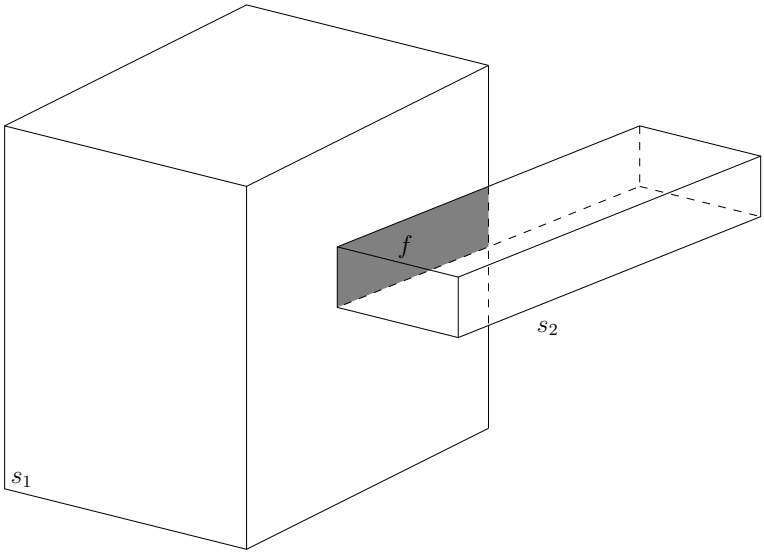



Figure 4.18 – Singularité résultant de l'application d'une opération booléenne à deux solides $s_1$ et $s_2$ représentés paramétriquement.

Notre première idée a été d'utiliser des librairies proposant une implémentation des opérations Booléennes (union, soustraction et intersection) pour les modèles 3D paramétriques, en particulier la librairie GTS (*GNU Triangulated Surface* [www]). Toutefois, la robustesse de la librairie GTS ne s'est pas montrée suffisante pour les besoins de notre application, une application successive des opérations Booléennes provoquant des erreurs liées à la représentation polyédriques des surfaces. Ces limitations nous ont amené à abandonner la représentation des volumes sémantiques à base de modèles à facettes pour envisager une représentation par surfaces implicites. L'implémentation des surfaces implicites ainsi que des opérations Booléennes nécessaires à notre méthode des volumes sémantiques a été réalisée en utilisant la bibliothèque VTK [www] (*The Visualization ToolKit*, développée par Kitware Inc. [SML98]). La suite de cette section est consacrée d'abord à une présentation succincte des concepts inhérents aux surfaces implicites et aux opérations correspondantes, puis à une description des étapes relatives à l'implémentation de la construction des volumes sémantiques.

### 4.5.1 Surfaces implicites

Les représentations purement géométriques des volumes 3D, *i.e.* la modélisation d'un volume par l'ensemble de ses frontières (*B-Rep representation* [Bau72, Bra75]) induisent une définition complexe des opérations Booléennes. De fait, afin d'implémenter l'intersection 3D des volumes sémantiques, nous proposons d'utiliser une représentation par surfaces implicites (ou plus exactement par volumes implicites) introduite par Wyvill et Wyvill[WW89]. Par abus de langage, nous parlons dans la suite de ce manuscrit des surfaces implicites comme de la méthode de modélisation de surfaces et volumes par des

fonctions plutôt que de la notion de fonction implicite en tant que telle.

**Définition 4.3** (fonction implicite)**.** Soit une fonction $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ et une constante $c \in \mathbb{R}$. La *surface implicite* associée à $f$ et $c$ est définie par l'ensemble :

$$S(f, c) = \{(x, y, z) \in \mathbb{R}^3 : f(x, y, z) = c\}. \tag{4.2}$$

**Définition 4.4** (volume implicite)**.** Soit une fonction $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ et une constante $c \in \mathbb{R}$. Le *volume implicite* associé à $f$ et $c$ est défini par l'ensemble :

$$V(f, c) = \{(x, y, z) \in \mathbb{R}^3 : f(x, y, z) \leq c\}. \tag{4.3}$$

Pour les surfaces implicites, chaque primitive $P_i$ est la source d'un champ de potentiel $F_i(x, y, z)$ également appelé fonction implicite. Pour tout point $P(x_P, y_P, z_P)$ de l'espace $\mathbb{R}^3$, la combinaison de différentes primitives $P_i$ correspondant chacune à une champ de potentiel $F_i$ par un opérateur d'agrégation permet d'évaluer l'ensemble des fonctions implicite au point $P$, c'est-à-dire :

$$F(x_P, y_P, z_P) = \mathcal{A}_i F_i(x_P, y_P, z_P)$$

L'opérateur $\mathcal{A}$ représentant un opérateur d'agrégation générique qui doit être instancié en fonction de la nature des relations entre les fonctions implicites que nous souhaitons modéliser (*e.g.* intersection, union, etc.).

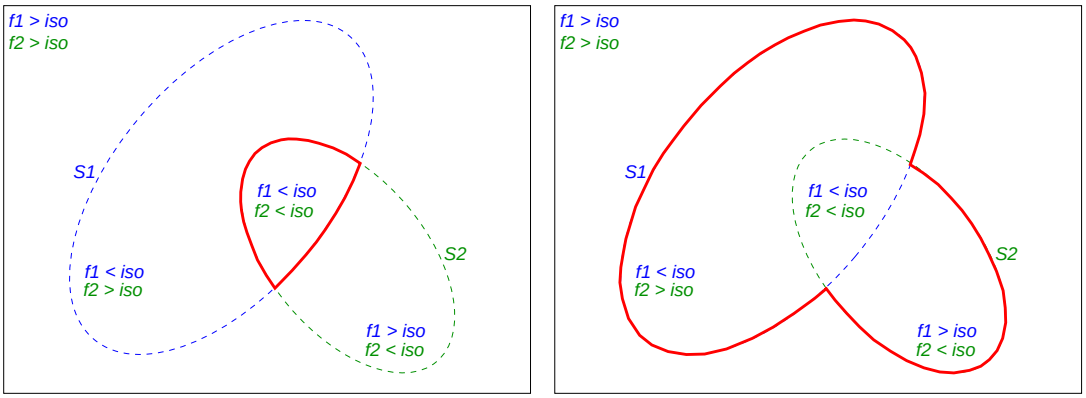
Les avantages liés à l'utilisation de fonctions implicites sont multiples :

- réalisation d'intersections, de soustractions et d'unions *exactes* de volumes 3D (contrairement aux techniques paramétriques qui génèrent des approximations dûes à l'imprécision de la représentation des surfaces) par composition des fonctions implicites,
- réalisation d'intersections, de soustractions et d'unions *robustes* (au contraire des opérations booléennes réalisées sur des modèles *B-Rep* qui sont sujettes aux singularités présentées brièvement ci-avant) par la conservation de la représentation implicite des surfaces ou des volumes,
- possibilité de tester de manière rapide et simple l'appartenance d'un point à un volume (ou une surface) par simple évaluation de la fonction implicite en ce point.

La librairie VTK fournit les opérations nécessaires à la création des volumes implicites ainsi que les opérations Booléennes associées. Toutefois, il n'existe pas de méthode générale permettant de déterminer le nombre de composantes connexes liées à une fonction implicite $f$. Ce nombre de composantes connexes nous est pourtant primordial puisqu'il correspond au nombre de classes de solutions du problème de placement de caméra.

Une solution consiste à approximer la surface implicite par un maillage de polyèdres, il est alors possible de calculer le nombre de composantes connexes de cette polygonalisation (tesselation). Toutefois, l'imprécision dûe à la représentation d'une surface implicite par un modèle polyèdrique conduit inévitablement à l'approximation du nombre de composantes connexes de cette dernière. Transposés à la méthode des volumes sémantiques, les composantes connexes représentent les différentes classes de solutions du problème de placement de caméra. Ainsi, une approximation du nombre de composantes connexes de la surface implicite correspond à une incertitude sur le nombre de classes de solutions. Toutefois, il est possible d'évaluer l'erreur commise lors de la polygonalisation en réalisant une approximation fidèle mais coûteuse de la surface implicite.

L'inconvénient majeur lié à l'utilisation des surfaces implicites réside dans le coût associé au maillage (ou tesselation, cf. [WW15]) des surfaces implicites. La tesselation est réalisée dans VTK par un algorithme de *Marching Cubes* [LC87] introduit par W.H. Lorensen et H.E. Cline lors de la conférence SIGGRAPH




(a) Intersection de surfaces implicites (b) Union de surfaces implicites

Figure 4.19 – Illustration de l'union et de l'intersection de deux surfaces ($S_1$ et $S_2$) définies par les fonctions de potentiel $f_1$ et $f_2$. Les équations des surfaces résultats des opérations Booléennes sont : $min(f_1, f_2) = iso$ (a), qui représente l'intersection des deux champs de potentiels; et $max(f_1, f_2) = iso$ (b) qui représente leur union.

en 1987. Cet algorithme permet de reconstruire les surfaces implicites en 3D en créant le maillage associé, en fonction d'une résolution qui va servir à l'échantillonnage de l'espace. Le principe du *Marching Cubes* est de recouvrir l'espace de recherche par des cubes unitaires (fonctions de la résolution fournie en paramètre) et d'inférer la configuration de la surface reconstruite en fonction de la position de chacun des sommets du cube par rapport à la surface (le sommet est-il devant, derrière ou sur la surface, cf. figure 4.20).

Cet algorithme est de complexité cubique en fonction de la résolution $n$ donnée ($O(n^3)$) et sera donc en pratique très lent pour reconstruire les volumes si nous souhaitons obtenir une approximation fidèle à la surface (la résolution doit alors être élevée). Cependant, les différentes dimensions du problème correspondent à des grandeurs physiques (position 3D de la caméra dans la scène), il est donc facile de déterminer la résolution à utiliser afin de minimiser l'erreur commise lors de l'approximation des volumes sémantiques. L'approximation des surfaces, bien que préjudiciable à la correction de l'approche des volumes sémantique, peut donc être minimisée et évaluée en fonction de la scène d'intérêt.

### 4.5.2 Description d'un problème de placement de caméra

L'ensemble des propriétés décrites dans la section 4.2 est proposé aux utilisateurs *via* un langage de script défini comme une extension du langage *Tcl/Tk* [www]. Cette solution est à la fois facile à appréhender pour l'utilisateur et peut être mise en œuvre de manière simple et rapide. Les propriétés sont simples à définir et le langage de script convient particulièrement bien aux tâches de tests de l'application ainsi qu'à l'évaluation des solutions fournies aux utilisateurs. Enfin, l'utilisation d'un tel type de langage permet une insertion aisée de notre approche dans les modeleurs 3D existants en assurant le développement de greffons spécialisés et en facilitant l'intégration à des interfaces utilisateur.

La table 4.1 met en correspondance les propriétés et les commandes du langage de script. Comme nous l'avons signalé en section 4.2.5, la propriété de cadrage prend en paramètres les coordonnées des

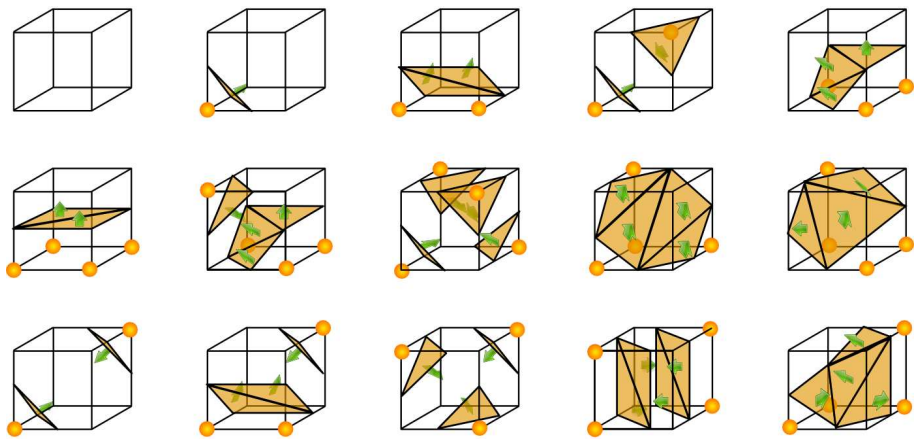


Figure 4.20 – Principe de l'algorithme du *Marching Cubes*. Le maillage est calculé en fonction des positions d'un cube unitaire par rapport à la surface que l'on souhaite reconstruire.

| Propriété | Script |
|---|---|
| Cadrage | `Frame Obj xmin xmax ymin ymax` |
| Orientation | `Orientation Obj viewing_angle` |
| Projection | `Projection Obj proj_distance` |
| Occlusion | `Occlusion Occder Occded type` |
| Absence d'occlusion | `NoOcclusion Obj` |
| Apparaître à l'écran | `SurEcran Obj` |
| Être à gauche de | `EstAGaucheDe Obj1 Obj2` |
| Être à droite de | `EstADroiteDe Obj1 Obj2` |

Table 4.1 – Langage de script associé aux différentes propriétés cinématographiques.

coins bas gauche et haut droit de la *frame* contenant l'objet fourni en second paramètre. Les coordonnées sont exprimées en espace écran et, de ce fait, des valeurs inférieures à $-1$ ou supérieures à 1 permettent de spécifier des cadres d'objets respectivement partiellement ou totalement hors-champ.

La propriété d'orientation nécessite la spécification d'un objet et d'un angle de vue parmi ceux présentés dans la sous-section 4.2.2. Afin de définir un plan de vue attaché à un objet, l'utilisateur doit caractériser la propriété de projection par un objet d'intérêt associé à un type de plan de vue (cf. sous-section 4.2.1).

Les propriétés d'occlusion associent, quant à elles, des objets occultants (`Occder`) à des objets occultés (`Occded`), ainsi que le type d'occlusion souhaité (totale, partielle ou absence d'occlusion). Enfin, les contraintes de positionnement relatifs à l'écran sont définies entre deux objets.

### 4.5.3 Des propriétés aux volumes sémantiques

Cette sous-section est consacrée à l'expression des propriétés cinématographiques en volumes sémantiques et à leur implémentation grâce à la bibliothèque VTK. Rappelons que les fonctions de coût

correspondantes à chaque propriété et nécessaires lors du processus de détermination du meilleur représentant ont été définies dans le chapitre 3.

#### *4.5.3.1 Propriété de projection*

Pour mettre en œuvre cette propriété, il nous faut définir des coefficients multiplicateurs à appliquer à chacun des différents plans de vue (très gros plan, gros plan, etc.) en fonction de la taille de l'englobant de l'objet.

Ensuite, nous devons nous assurer que la taille de l'objet à l'écran soit au minimum celle calculée en fonction de la taille de son englobant et du facteur multiplicatif correspondant au plan de vue choisi par l'utilisateur. Cela revient à assurer que la projection d'un carré de coté au égal à la taille minimale sur une des dimensions de l'englobant de l'objet soit cohérente avec le type de projection désiré.

Nous devons calculer les matrices de projection nécessaires pour limiter les positions possibles de la caméra. Nous présentons les matrices $R_y$ et $R_z$ de rotation d'angles $\theta$ et $\phi$ autour des axes $OY$ et $OZ$, ainsi que la matrice $T$ de translation nécessaires au calcul du projeté d'un point 3D $P(X, Y, Z)$ sur l'écran.

$$T = \begin{pmatrix} 1 & 0 & 0 & (X - X_C) \\ 0 & 1 & 0 & (Y - Y_C) \\ 0 & 0 & 1 & (Z - Z_C) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos\phi & \sin\phi & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Le point $P'(t) \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix}$ transformé de $P$ dans le repère de la caméra est alors défini par :

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{pmatrix} (X - X_C)\cos\theta\cos\phi - (Y - Y_C)\cos\theta\sin\phi + (Z - Z)\sin\theta \\ (X - X_C)\sin\phi + (Y - Y_C)\cos\phi \\ -(X - X_C)\sin\theta\cos\phi + (Y - Y_C)\sin\theta\cos\phi + (Z - Z_C)\cos\theta \end{pmatrix}$$

Puis $P''$ la projection de $P'$ sur l'écran :

$$P'' \begin{pmatrix} X'' \\ Y'' \end{pmatrix} = \begin{pmatrix} \frac{X'}{Z'/\gamma} \\ \frac{Y'}{Z'/\gamma} \end{pmatrix}$$

où $\gamma$ représente la distance focale de la caméra filmant la scène.

En pratique nous remplacons les coordonnées du point $P(X, Y, Z)$ par un intervalle de valeurs. Pour ce faire, nous multiplions la valeur du rayon de la sphère englobante de l'objet par le coefficient correspondant au type de plan de vue spécifié pour la propriété. Cet intervalle nous assure que la projection de l'objet est contenue entièrement dans le cadre et évite ainsi un respect partiel des propriétés de *framing*.

Nous ajoutons à la valeur obtenue, qui correspond à la position idéale de caméra, un epsilon autorisant une variation autour de celle-ci, afin de pouvoir définir deux valeurs : $R_{min}$ et $R_{max}$ qui sont les rayons intérieurs et extérieurs de la sphère creuse représentant la région des positions cohérentes de caméra par rapport à la propriété spécifiée. Nous obtenons ainsi des volumes similaires à ceux présentés en figure 4.6 page 102.

Il est important de noter que les valeurs de distance focale obtenues à l'issue du processus de résolution vont avoir un impact très important sur l'aspect visuel de l'image résultat. Il est donc primordial de choisir avec précaution le domaine des valeurs associées à la variable modélisant la distance focale $\gamma$, sous peine d'obtenir des images fortement distordues.

#### *4.5.3.2 Propriété d'orientation*

La propriété d'orientation est définie par un vecteur orienté en fonction de l'orientation intrinsèque des objet de la scène. Chaque orientation proposée pour cette propriété (profil droit, vue de face, etc.) permet d'établir un angle de vue idéal entre la caméra et l'objet. Ici encore, ce résultat exact est modulé pour ajouter de la flexibilité au système, comme illustré en figure 4.21.

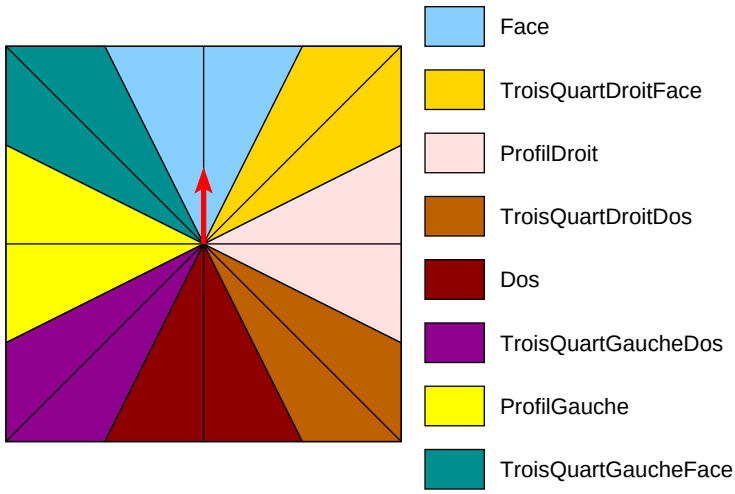



Figure 4.21 – Variations acceptées par rapport au vecteur d'orientation intrinsèque d'un objet.

Le calcul de l'orientation idéale en fonction du vecteur d'orientation intrinsèque d'un objet ($\overrightarrow{V_{\text{obj}}}$) est basé sur la création d'une base orthonormée à partir de ce vecteur. Nous devons déterminer deux vecteurs orthogonaux à $\overrightarrow{V_{\text{obj}}}$. Pour ce faire, nous appliquons la méthode suivante :

Soit un vecteur $\overrightarrow{V}(v_X, v_Y, v_Z)$, le vecteur $\overrightarrow{V_\perp}$ orthogonal à $\overrightarrow{V}$ est obtenu directement à partir des coordonnées de ce dernier :

$$\overrightarrow{V_\perp} = (-v_Y, v_X, 0)$$

Un cas particulier reste toutefois à prendre en compte, *i.e.* lorsque $v_Y = v_X = 0$ (dans ce cas $\overrightarrow{V_\perp}$ est instancié au vecteur nul). La solution consiste à tester si $v_Y = 0$ et à le remplacer par 1 si tel est le cas. Le vecteur $V_\perp$ devient alors :

$$\overrightarrow{V_\perp} = (-1, 0, 0)$$

Soit $\perp$ l'opérateur calculant le vecteur $\overrightarrow{V_\perp}$ en fonction d'un vecteur $\overrightarrow{V}$ de l'espace tridimensionnel.

Le dernier vecteur de la base orthnormée s'obtient naturellement en réalisant le produit vectoriel de $\overrightarrow{V}$ par $\overrightarrow{V_\perp}$, *i.e.* $\overrightarrow{V_{\perp\perp}} = \overrightarrow{V} \wedge \overrightarrow{V_\perp}$ (où $\wedge$ représente l'opérateur de produit vectoriel). En simplifiant l'écriture, le système de coordonnées sera défini par :

$$\mathcal{B}_{obj} = \langle \overrightarrow{V_{X_{\text{obj}}}}, \overrightarrow{V_{Y_{\text{obj}}}}, \overrightarrow{V_{Z_{\text{obj}}}} \rangle$$

avec

$$\begin{array}{rcl} \overrightarrow{V_{X_{\text{obj}}}} & = & \overrightarrow{V_{\text{obj}}} \\ \overrightarrow{V_{Y_{\text{obj}}}} & = & \perp (\overrightarrow{V_{X_{\text{obj}}}}) \\ \overrightarrow{V_{Z_{\text{obj}}}} & = & \overrightarrow{V_{X_{\text{obj}}}} \wedge \overrightarrow{V_{Y_{\text{obj}}}} \end{array}$$

L'angle de vue souhaité par l'utilisateur ainsi que le vecteur d'orientation intrinsèque de l'objet permet de définir un angle $\alpha$ correspondant à une rotation des vecteurs $\overrightarrow{V_{X_{\text{obj}}}}$ et $\overrightarrow{V_{Z_{\text{obj}}}}$ de la base orthonormée $\mathcal{B}_{obj}$ calculée précédement. Le vecteur orientation depuis lequel la caméra doit filmer l'objet est calculé comme suit :

$$\overrightarrow{I_{\text{cam}}} = -(\overrightarrow{V_{X_{\text{obj}}}} * \cos(\alpha) + \overrightarrow{V_{Z_{\text{obj}}}} * \sin(\alpha))$$

Afin d'éliminer les positions de caméra non cohérentes, il faut minimiser la différence entre $\overrightarrow{I_{\text{cam}}}$ et $\overrightarrow{V_{\text{obj}}}$. Pour ce faire, nous devons transformer les angles d'Euler de la caméra (*i.e.* $\phi, \theta, \psi$ exprimés en radians) en leurs équivalents vectoriels, c'est-à-dire transformer une orientation angulaire en orientation vectorielle. Cette transformation est donnée par la relation suivante entre $(\theta_C, \phi_C, \psi_C)$ et leurs équivalents $(V_{X_{\text{C}}}, V_{Y_{\text{C}}}, V_{Z_{\text{C}}})$ :

$$\begin{array}{rcl} V_{X_C} & = & \cos(\phi_C) * \cos(\theta_C + \frac{\pi}{2}) \\ V_{Y_C} & = & \sin(\phi_C) \\ V_{Z_C} & = & \cos(\phi_C) * \sin(\theta_C + \frac{\pi}{2}) \end{array}$$

#### *4.5.3.3 Propriété d'occlusion*

La propriété d'occlusion est basée sur la création des cônes d'occlusion partielle et totale (cf. sous-section 4.2.3). Leur modélisation est possible grâce à (ou à cause de) la représentation des objets que nous avons adopté, c'est-à-dire une modélisation par sphères englobantes, cf. annexe A.0.2. Nous devons être capable de modéliser à la fois une occlusion partielle, une occlusion totale et une absence d'occlusion entre deux objets. Ces différentes classes d'occlusion donnent lieu à la génération de trois types de cônes différents, dont la construction sera traitée dans les paragraphes suivants.

**Cônes d'occlusion partielle** La figure 4.22 illustre les concepts sous-jacents à la conception de ces cônes, à partir des sphères englobantes de deux objets $A$ et $B$ d'une scène 3D.

L'idée première est de définir les plans tangents aux deux sphères englobantes $S_A$ et $S_B$, de centre $A$ et $B$ représentant respectivement les objets $A$ et $B$, de telle manière que ces plans se croisent et se coupent en une droite $\mathcal{I}$. Le point d'intersection Inter de la figure 4.22 est ensuite obtenu comme étant l'intersection entre de la droite $\mathcal{I}$ et du vecteur reliant les centres des deux objets ($\overrightarrow{V_{\text{obj}}}$). La position de ce point est calculée en fonction des rayons ($R_A$ et $R_B$) des sphères englobantes $S_A$ et $S_B$, du vecteur normé reliant les centres des deux objets ($\overrightarrow{V_{\text{obj}}}$), ainsi que des distances indiquées sur la figure. Nous devons tout

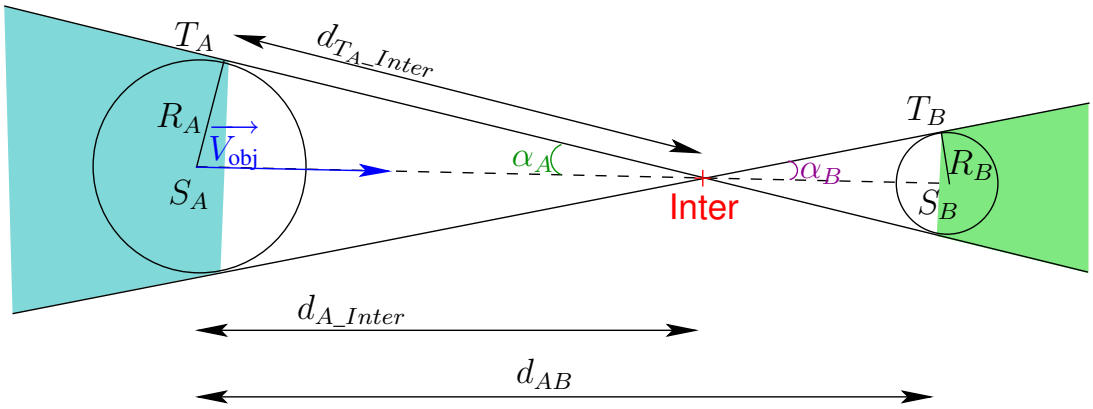



Figure 4.22 – Cônes d'occlusion partielle : les zones mises en évidence correspondent aux régions d'occlusion partielle d'un objet par un autre.

d'abord calculer la distance entre le point d'intersection et l'objet $A$, ceci s'effectue en appliquant des notions trigonométriques de base :

$$d_{A\_Inter} = d_{AB} - (\frac{d_{AB}}{\frac{R_A}{R_B} + 1})$$

Grâce à cette distance, nous déterminons les coordonnées du point Inter (aligné avec $A$ et $B$, il se trouve donc le long du vecteur $\overrightarrow{V_{\text{obj}}}$) :

$$\text{Inter} = A + \overrightarrow{V_{\text{obj}}} * d_{A\_Inter}$$

Nous possédons ainsi toutes les informations nécessaires pour calculer les valeurs manquantes pour la construction des cônes d'occlusion partielle:

$$\begin{aligned}
d_{Tangent_A_Inter} &= \sqrt{{d_{A_Inter}}^2 - {R_A}^2} \\
\alpha_A &= \arccos(\frac{d_{Tangent_Inter}}{d_{A_Inter}}) \\
d_{B_Inter} &= d_{AB} - d_{A_Inter} \\
d_{Tangent_B_Inter} &= \sqrt{{d_{B_Inter}}^2 - {R_B}^2} \\
\alpha_B &= \arccos \frac{d_{Tangent_B_Inter}}{d_{B_Inter}}
\end{aligned}$$

Nous avons calculé tous les paramètres nécessaires à la construction des deux cônes d'occlusion partielle en tant que surfaces implicites dans VTK. En effet, il suffit de spécifier un angle de rotation (en degrés) pour créer un cône, ensuite nous devons calculer les transformations nécessaires à son bon positionnement (centré sur les objets et orienté correctement). Ces transformations sont calculées à partir du vecteur $\overrightarrow{V_{\text{obj}}}$, de son opposé $\overrightarrow{V_{\text{objopp}}}$ $(= -\overrightarrow{V_{\text{obj}}})$ et des coordonnées du point d'intersection Inter.

La construction des cônes d'occlusion partielle par le biais de la librairie VTK requiert des orientations spécifiées en coordonnées sphériques, *i.e.* trois coordonnées $r, \theta, \phi$. Ainsi nous devons transformer les composantes des vecteurs $\overrightarrow{V_{\text{obj}}}$ et $\overrightarrow{V_{\text{objopp}}}$ en coordonnées sphériques. Ceci s'effectue par l'application

des relations trigonométriques suivantes, dans lesquelles $V_X, V_Y$ et $V_Z$ représentent respectivement les composantes $X$, $Y$ et $Z$ du vecteur d'intérêt :

$$\begin{aligned} r &= \sqrt{{V_X}^2 + {V_Y}^2 + {V_Z}^2} \\ \phi &= \arcsin(\tfrac{V_Z}{r}) \\ \theta &= \arctan(V_Y, V_X) \end{aligned}$$

Nous obtenons les valeurs $(r_{\text{obj}}, \phi_{\text{obj}}, \theta_{\text{obj}})$ d'un côté et $(r_{\text{objopp}}, \phi_{\text{objopp}}, \theta_{\text{objopp}})$ d'autre part, en appliquant ces relations à $\overrightarrow{V_{\text{obj}}}$ et $\overrightarrow{V_{\text{objopp}}}$. Les coordonnées sphériques nous fournissent les angles nécessaires aux rotations et à la translation des cônes $C_A$ et $C_B$. La dernière étape de la construction des cônes consiste en la définition d'une matrice de transformation dans VTK et en son application aux deux cônes formés à partir des angles $\alpha_A$ et $\alpha_B$. Enfin, pour créer les surfaces implicites finales à partir de ces deux cônes, nous en réalisons l'union Booléenne, comme présenté en figure 4.10.

**Cônes d'occlusion totale** Concernant les cônes d'occlusion totale, une approche similaire est adoptée. Les plans que nous cherchons à construire sont ceux identifiés sur la figure 4.23. Grâce à ce schéma, nous identifions clairement deux cas de figure liés à la construction de ces cônes :

- les deux sphères ont un rayon identique : dans ce cas nous ne parlons pas de cônes puisque les plans illustrés sur la figure 4.23 ne se coupent pas, mais d'un cylindre d'occlusion.
- les sphères sont de rayons différents : le point d'intersection Inter n'est plus situé entre les deux objets $A$ et $B$ mais derrière l'un d'eux (celui dont la sphère sera la plus petite). Cependant, la méthode de construction reste identique à celle des cônes d'occlusion partielle.

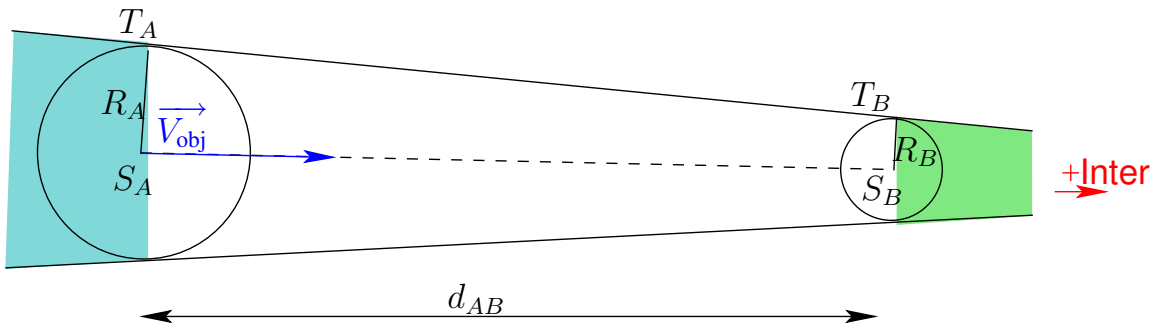



Figure 4.23 – Cônes d'occlusion totale.

**Absence d'occlusion** Modéliser une absence d'occlusion s'avère très facile par notre méthode. En effet, il suffit de construire un englobant général de la scène 3D, puis de construire les cônes d'occlusion partielle des deux objets impliqués dans la propriété d'absence d'occlusion. Enfin, nous appliquons une opération de différence Booléenne entre la surface implicite modélisant l'englobant de la scène et les cônes d'occlusion partielle.

Le volume résultant de cette différence représente les zones de l'espace de recherche pour lesquelles les objets impliqués dans la propriété ne s'occultent pas l'un l'autre. Nous pouvons donc représenter l'absence d'occlusion entre deux objets comme étant la négation de la propriété d'occlusion partielle calculée sur ces mêmes objets.

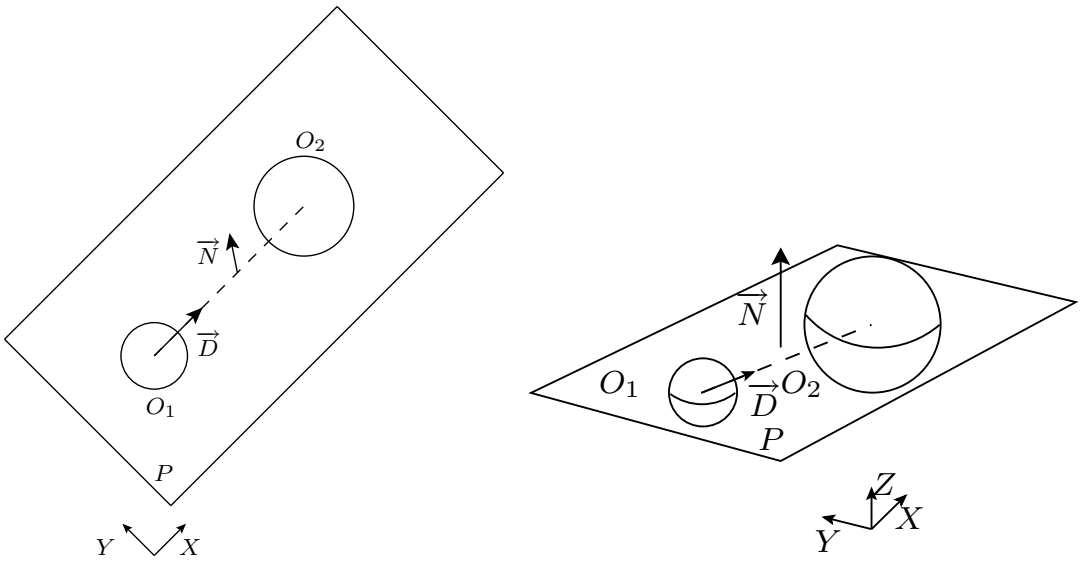
#### *4.5.3.4 Propriété de cadrage*

L'implémentation de la propriété de cadrage consiste en la donnée de deux fonctions objectif qui expriment (*i*) la projection de l'objet d'intérêt au milieu du cadre défini par l'utilisateur et (*ii*) la maximisation de la taille de projection de l'objet à l'intérieur de ce cadre (*i.e.* maximiser la taille du projeté de l'objet par rapport au plus petit côté du cadre spécifié).

La première partie de la contrainte s'apparente donc à une contrainte de projection simple, alors que la deuxième partie est en partie dérivée de la contrainte de projection.

#### *4.5.3.5 Propriété de positionnement relatif*

Cette propriété a pour but de placer à l'écran deux objets l'un par rapport à l'autre et est basée sur la découpe de l'espace en fonction de plans de coupe. Toutefois, comme nous travaillons en 3D, ceux-ci doivent être définis par rapport à un vecteur normal.




(a) Plan de découpe en vue de dessus.

(b) Vue de côté du plan de découpe

Figure 4.24 – Plan de découpe lié à la propriété de positionnement relatif gauche/droite à l'écran.

**Exemple 4.7 (Positonnement relatif « Gauche/Droite » à l'écran).** *Pour une propriété de positionnement relatif « Gauche/Droite » à l'écran entre deux objets $A$ et $B$, nous définissons le plan de coupe (cf. figure 4.24) en utilisant le vecteur propre de l'objet $A$ comme vecteur normal de ce plan. Si nous utilisons une représentation paramétrique d'un plan :*

$$(P) : ax + by + cz + d = 0$$

*Les coefficients $a, b$ et $c$ sont remplacés par les composantes du vecteur propre $V_A(V_X, V_Y, V_Z)$, nous avons donc la définition suivante du plan $P$ :*

$$(P) : V_X x + V_Y y + V_Z z + d = 0 \tag{4.4}$$

*La valeur $d$ est quant à elle calculée en résolvant l'équation 4.4 avec pour valeurs $(x, y, z)$ les coordonnées du centre de l'objet $A$ qui appartient au plan $P$.*

### 4.5.4 Calcul des meilleurs représentants

Le calcul des meilleurs représentants consiste à calculer une configuration de caméra à l'intérieur de chacun des volumes sémantiques solutions du problème pour laquelle son orientation et sa distance focale satisfont toutes les contraintes. En effet, les volumes sémantiques permettent de discriminer les positions de caméra consistantes de celles ne vérifiant pas les propriétés utilisateur, ainsi les paramètres $(X_C, Y_C, Z_C)$ (cf. annexe A.0.1) sont instanciés en testant l'appartenance d'une configuration à un volume sémantique solution.

Le calcul d'une configuration cohérente s'effectue lors du processus numérique par application d'une procédure de Recherche Locale Continue (RTC, cf. section 3.3.1). Chaque propriété cinématographique possède une fonction objectif (ou de pénalité) représentant l'adéquation d'une configuration de caméra à la satisfaction de cette propriété. La fonction objectif est minimale lorsqu'une configuration de caméra satisfait le mieux possible la propriété correspondante. Par exemple, comme nous l'avons présenté en section 4.3.3, la fonction de pénalité associée à la propriété : « Voir un objet de profil » s'annule pour toutes les configurations de caméra appartenant exactement à une droit orthogonale au vecteur vision de l'objet d'intérêt, cf. figure 4.17.

Le principe de l'algorithme de Recherche Locale Continue est de combiner les fonctions objectif de chacune des propriétés impliquées dans la description utilisateur en une seule fonction de coût globale au problème de placement de caméra. Comme nous l'avons présenté au chapitre 3, section 3.3.2, l'agrégation des différentes fonctions objectif est problématique. En effet, les opérateurs utilisés pour ce regroupement de fonctions doivent être choisis avec précaution afin de ne pas dénaturer la fonction de coût finale, c'est-à-dire que toutes les fonctions objectif représentent la même contribution pour l'évaluation des configurations de caméra. De même, la définition des fonctions objectif doit être effectuée avec prudence, les évaluations de ces dernières devant être comparables et agrégables entre elles sous peine de fausser la fonction de coût finale.

À l'issue de cette étape, nous sommes en mesure de présenter à l'utilisateur un bon représentant, si il existe, *i.e.* une configuration de caméra satisfaisant au mieux les propriétés du problème initial, pour chacun des volumes sémantiques solutions, c'est-à-dire pour chaque classe de solution du problème de placement de caméra.

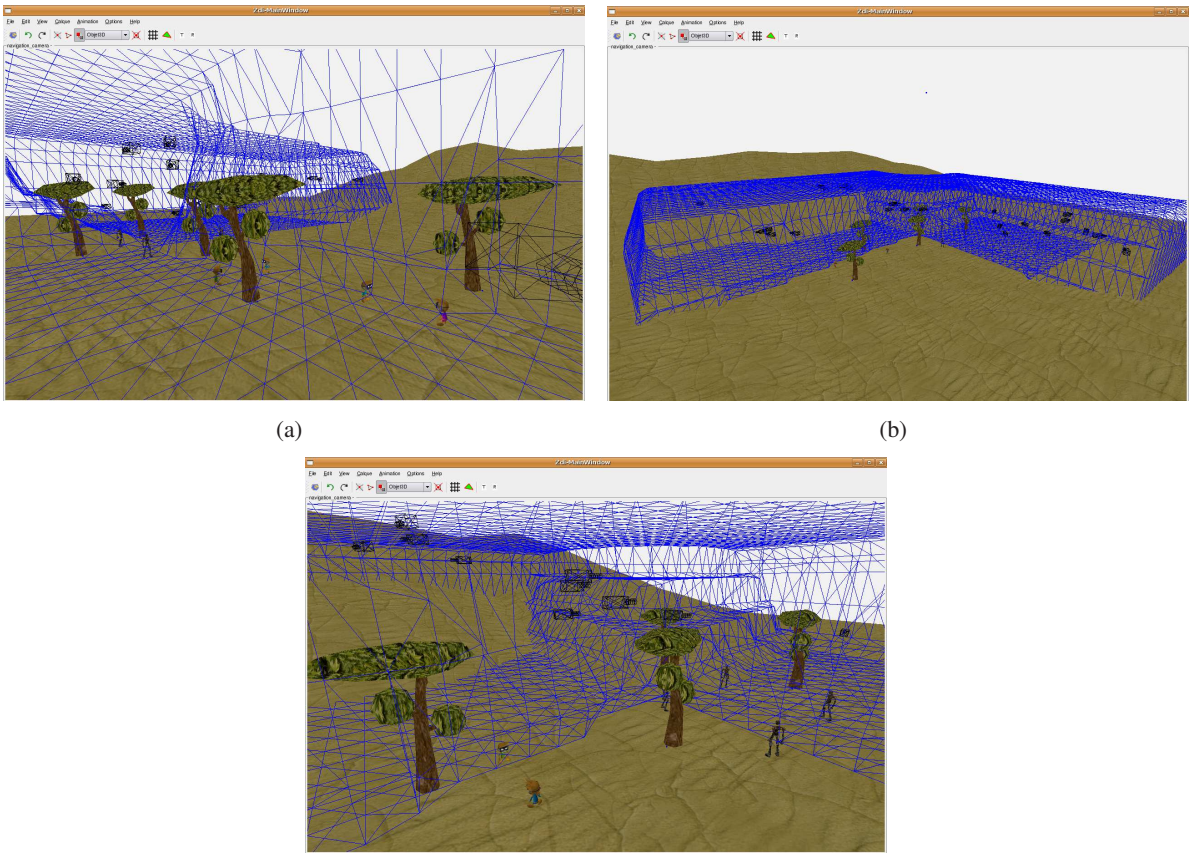
### 4.5.5 Présentation des résultats à l'utilisateur

La particularité de l'approche des volumes sémantiques est, contrairement aux approches existantes, de présenter l'ensemble des classes de solution équivalentes à un problème de placement de caméra en environnements virtuels. Pour ce faire, nous devons être capables d'identifier les différents volumes représentant chaque classe de solutions et de présenter pour chacun d'eux une configuration de caméra caractéristique.

L'approche choisie est d'afficher, au sein du modeleur ZDM (cf. annexe B), des polygones en représentation « fil de fer » (*wire frame*) pour chacun des volumes sémantiques directement dans la scène 3D étudiée. Les volumes sont regroupés au sein d'un calque ZDM permettant à l'utilisateur de choisir si il souhaite ou non afficher l'information correspondante. De plus, un système de caméras multiples a été implémenté au sein de ZDM permettant ainsi à l'utilisateur de naviguer entre les différentes configura-

tions caractéristiques de chacun des volumes sémantiques solution. Ainsi, il est possible de comparer les différences obtenues pour chaque caméra représentative des classes de solution en termes de propriétés à l'écran afin que l'utilisateur puisse choisir celle qui lui convient le mieux.

La décision d'afficher les volumes en fil de fer et de permettre à l'utilisateur de naviguer entre les différentes caméras caractéristiques de chacun d'eux permet à la fois de visualiser rapidement l'existence des classes de solution du problème et d'avoir un aperçu concret des différences existant entre les différents volumes sémantiques satisfaisant la description utilisateur.



(a) (b)

(c)

Figure 4.25 – Présentation des volumes sémantiques issus d'une description utilisateur dans le logiciel ZDM. Un ensemble de caméras solutions calculées lors du procéssus numérique sont affichées à l'intérieur de ce volume. L'utilisateur peut ensuite naviguer entre les différentes caméras afin de visualiser les différents résultats.

La figure 4.25 présente des captures d'écran du logiciel ZDM illustrant la présentation des caméras[1] issues du processus de résolution numérique à l'intérieur d'un volume sémantique résultat d'une description utilisateur.

[1] N.B. : seules les positions des caméras illustrées sont correctes, les orientations ne le sont pas dans ZDM.

## 4.6 Résultats

Afin d'illustrer l'approche des volumes sémantiques, nous proposons maintenant trois exemples basés sur les propriétés cinématographiques définies dans ce chapitre. Le premier exemple reproduit un placement de caméra caractéristique de la cinématographie : l'*over-the-shoulder*.

Le deuxième exemple illustre les capacités de caractérisation des solutions pour un problème de composition visuelle dans une image. L'utilisateur spécifie l'agencement de cinq objets dans une image (*five frames shot*), la description correspondante aboutissant à l'existence de différentes classes de solution au problème posé. Nous réutilisons ensuite ce même exemple afin d'illustrer les possibilités de raffinement d'un problème au vu des solutions obtenues. Dans ce troisième exemple, après avoir été informé de l'existence de plusieurs classes de solution au problème de *five frames shot*, l'utilisateur souhaite ajouter des propriétés supplémentaires à sa description initiale, afin d'obtenir la plan de vue désiré.

Enfin, un dernier exemple présente les limites de notre approche lorsque de trop nombreuses occlusions doivent être prises en compte pour un problème de placement de caméra en environnement virtuel. En effet, les occlusions sont sources de création de volumes sémantiques distincts augmentant ainsi considérablement le coût de la polygonalisation de la surface implicite modélisant les solutions du problème. Ces problèmes peuvent apparaître dans des scènes complexes comportant de nombreux objets pour lesquels l'utilisateur souhaite éviter les occlusions à l'écran.

### 4.6.1 L'*over-the-shoulder*

Notre premier exemple consiste en la modélisation, par l'approche des volumes sémantiques, d'un plan de vue classique en cinématographie : l'*over-the-shoulder*. Celui-ci est traditionnellement utilisé lors du dialogue de deux acteurs. Il consiste à placer la caméra derrière l'épaule d'un des protagonistes, afin de filmer à la fois l'épaule et le dos de la tête de l'acteur qui parle et le visage de son interlocuteur, permettant ainsi au spectateur d'étudier les réactions de celui-ci en fonction du discours.

La modélisation de ce probème de placement de caméra comporte deux objets $A$ et $B$, représentant les deux interlocuteurs de la scène. Les propriétés cinématographiques necéssaires à la description d'un *over-the-shoulder* sont présentées dans le script suivant :

```
#voir A à gauche de l'image à n'importe quelle hauteur
Frame $objA -0.8 -0.3 -1 1
#voir B au milieu et à droite de l'image
Frame $objB 0.3 0.8 -0.5 0.5
Orientation $objA TroisQuartDroitDos
Orientation $objB Face
Projection $objA PleinCadre
Projection $objB PlanEnsemble
NoOcclusion $objA $objB
```

Ce problème de placement de caméra aboutit à la découpe de l'espace de recherche illustrée en figure 4.26. La création des volumes sémantiques associés à ces propriétés est effectuée par l'opérateur $G_f$ de filtrage géométrique. Le résultat de l'application de cet opérateur aux différentes propriétés est illustré en figure 4.27.

Le volume sémantique correspondant à la description utilisateur (illustré en figure 4.28) est ensuite calculé par intersection des volumes sémantiques obtenus pour chacune des propriétés par le processus géométrique. Toutes les positions de caméra situées à l'intérieur de ce volume peuvent aboutir à l'obtention d'une image satisfaisant la description de l'*over-the-shoulder*, un exemple est présenté en

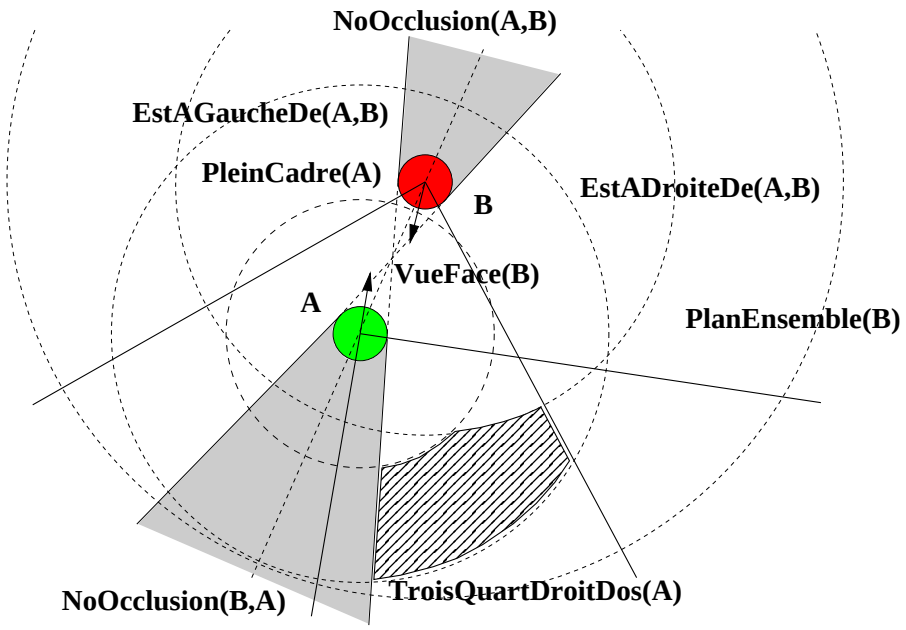



Figure 4.26 – Partitionnement sémantique de l'espace de recherche associé au problème de l'*over-the-shoulder*.

figure 4.29. Le processus numérique est ensuite appliqué à l'intérieur de ce volume afin de calculer une configuration (*i.e.* position et orientation de caméra) maximisant les fonctions objectif associées aux propriétés cinématographiques de la description utilisateur. Le meilleur candidat est ainsi calculé et le résultat obtenu depuis cette configuration de caméra peut être présenté à l'utilisateur. Une image produite dans le logiciel ZDM en utilisant le meilleur candidat calculé par le processus numérique de résolution pour l'*over-the-shoulder* est présentée en figure 4.30.

### 4.6.2 Positionnement de cinq objets dans une image : le *five frames shot*

Notre deuxième exemple consiste en la description d'un agencement de cinq objets coplanaires $A, B, C, D, E$ dans une image. L'utilisateur souhaite voir les objets $A, B$ et $C$ respectivement à gauche, au milieu et à droite de l'écran. Les deux objets $D$ et $E$ quant à eux doivent apparaître à l'écran sans être occultés. Le script correspondant est le suivant :

```
#voir A à gauche de l'image à n'importe quelle hauteur
Frame $objA -1.0 -0.33 -1.0 1.0
#voir B au milieu de l'image à n'importe quelle hauteur
Frame $objB -0.33 0.33 -1.0 1.0
#voir C à droite de l'image à n'importe quelle hauteur
Frame $objC 0.33 1.0 -1.0 1.0
#voir D à l'image à n'importe quelle hauteur
Frame $objD -1.0 1.0 -1.0 1.0
#voir E à l'image à n'importe quelle hauteur
Frame $objE -1.0 1.0 -1.0 1.0
```

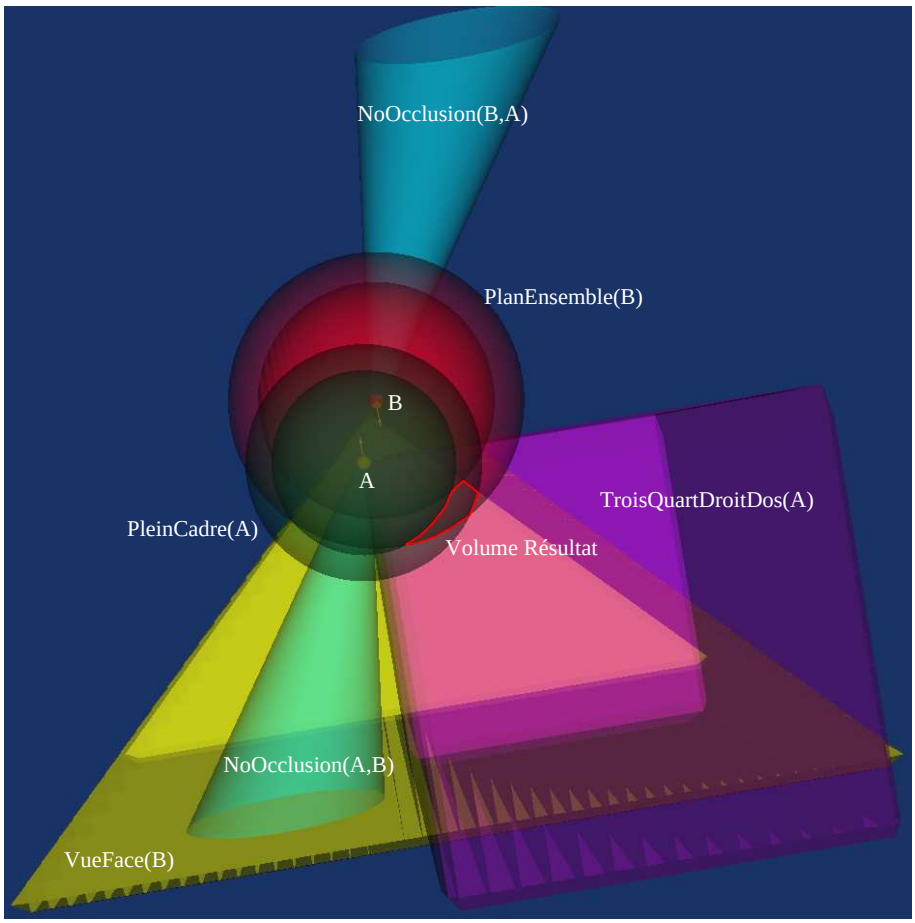



Figure 4.27 – Volumes sémantiques générés par le processus de partitionnement sémantique de l'espace de recherche et affichés par la librairie VTK. L'intersection de ces volumes correspond aux positions de caméra solutions de l'*over-the-shoulder*.

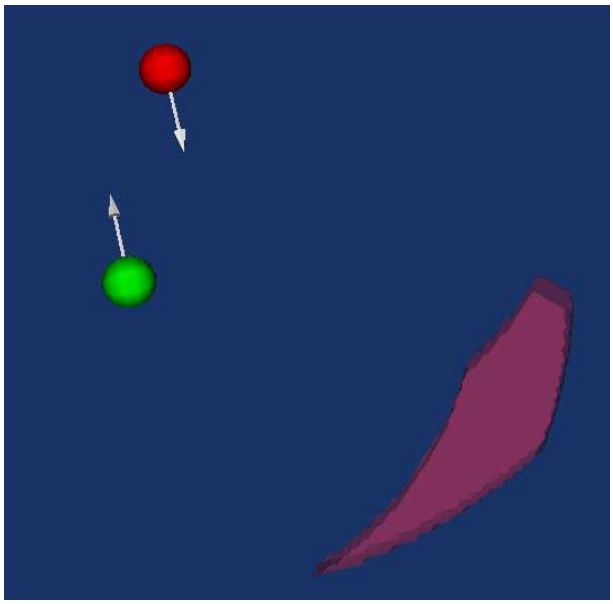


Figure 4.28 – Volume sémantique solution obtenu par intersection des volumes illustrés en figure 4.27.

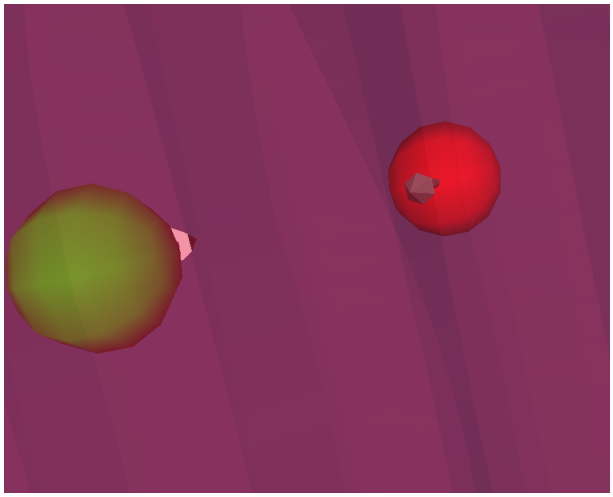


Figure 4.29 – Exemple de positionnement d'une caméra à l'intérieur du volume sémantique résultat calculé par le processus de partitionnement géométrique de l'espace et illustré en figure 4.28.


Figure 4.30 – Image produite par la configuration de caméra calculée à l'issue du processus numérique de résolution intégré au logiciel ZDM.

```
#pas d'occlusions pour D et E
NoOcclusion $objD
NoOcclusion $objE
```

La figure 4.31 illustre la modélisation 2D des propriétés définies par l'utilisateur. Les volumes sémantiques générés par l'opérateur de filtrage géométrique correspondants à l'intersection des différentes propriétés sont présentés en figure 4.32.

Dans cet exemple, l'existence de plusieurs régions non-connexes de l'espace indique qu'il existe plusieurs classes de solutions distinctes au problème d'agencement des cinq objets à l'image ou *five frames shot*. Chaque volume isolé amène à une prise de vue caractéristique par rapport à la description utilisateur. En effet, la position des objets $D$ et $E$ n'étant pas spécifiée à l'image, les différents volumes vont représenter des classes de solutions distinctes vis-à-vis de la position de ces deux objets à l'écran.

L'image supérieure de la figure 4.33 présente les différents volumes sémantiques solutions au *five frames shot* affichés en tant que fil de fer au sein de la scène 3D étudiée. Les trois captures d'écran inférieures illustrent quant à elles les meilleurs représentants de trois volumes sémantiques obtenus lors du processus numérique, *i.e.* de recherche locale continue. L'image en bas à gauche propose une vue caractérisant le volume sémantique dans lequel l'objet $D$ est situé à gauche de l'objet $A$ et l'objet $E$ est situé entre $A$ et $B$. La capture d'écran médiane correspond à la meilleure configuration de caméra issue du volume sémantique central, lequel est caractérisé par une disposition des objets $D$ et $E$ de part et d'autre de l'objet $B$. Enfin, l'image de droite a été obtenue à partir du volume sémantique englobant les positions de caméra aboutissant à la composition visuelle suivante : l'objet $E$ est situé à la droite de $C$ et l'objet $D$ est situé entre les objets $B$ et $C$.

Cet exemple illustre clairement les limitations des approches existantes de placement de caméra. En effet, ces dernières ne présentant qu'une seule solution à l'utilisateur, sans lui laisser le choix entre les différentes configurations équivalentes au vu de la description utilisateur. La méthode des volumes sémantiques permet de présenter les classes de solution et illustre chacune d'elle par une prise de vue caractéristique. Ainsi, l'utilisateur peut choisir la caméra qui lui convient le mieux.

Cet exemple permet également de présenter une autre spécificité de notre approche, le raffinement

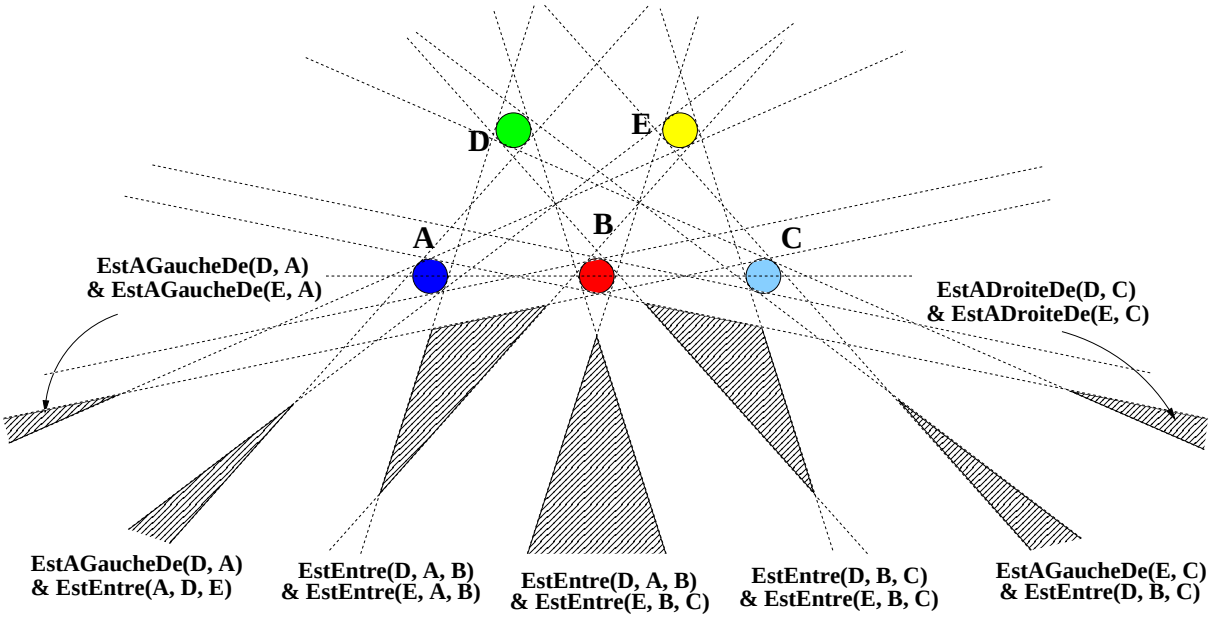



Figure 4.31 – Modélisation de l'exemple *five frames shot*.

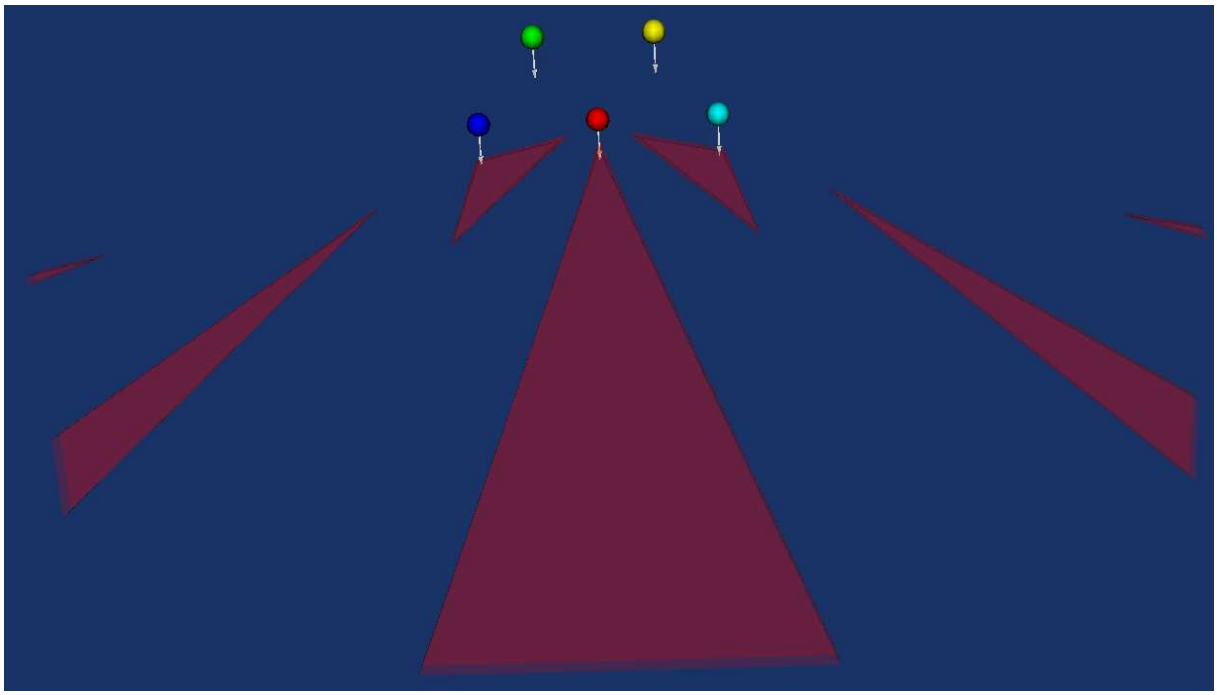


Figure 4.32 – Volumes sémantiques générés grâce à la description du *five frames shot*.

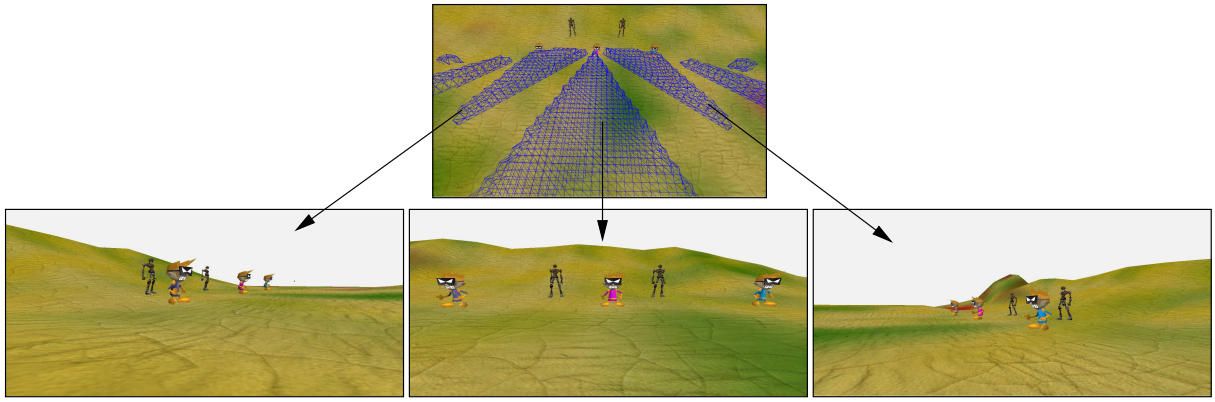


Figure 4.33 – Chaque classe de solutions correspond à un volume isolé et à un type de prise de vue caractéristique.

d'une description après avoir obtenu le résultat du problème initial. En effet, si l'utilisateur après avoir spécifié le script correspondant au *five frames shot* obtient les résultats présentés en figure4.32, il peut décider de caractériser plus finement ces volumes en réalisant une exploitation (cf. section 4.4), dont le résultat est présenté en figure 4.34.

La figure 4.35 présente un raffinement de la description du *five frames shot* pour laquelle les propriétés suivantes ont été ajoutées à la description initiale :

```
#voir A de face
VueFace $objA
#voir B de face
VueFace $objB
#voir C de face
VueFace $objC
```

Les volumes sémantiques correspondants sont obtenus par application de l'opérateur $G_f$ à ces trois nouvelles propriétés, puis par intersection Booléenne du résultat de ce filtrage géométrique avec les volumes solutions du *five frames shot*. Le volume résultat est présenté en figure 4.35, contient les positions de caméra satisfaisant les caractéristiques du problème initial (*five frames shot*), c'est-à-dire des configurations proposant des vues de face de chacun des objets $A$, $B$ et $C$.

### 4.6.3 Limites de l'approche proposée

Le dernier exemple que nous présentons dans ce manuscrit montre les limites de notre approche. En effet, comme nous l'avons décrit, le coût associé à la tesselation des volumes sémantiques obtenus lors du processus de filtrage géométrique représente le principal goulot d'étranglement de notre approche. De même, le processus de résolution numérique (c'est-à-dire la procédure de recherche locale continue) peut être rhédibitoire lorsque les paramètres ne sont pas bien choisis. Ainsi, nous souhaitons présenter un exemple démontrant les limites de l'approche des volumes sémantiques.

La description du problème consiste en une scène 3D composée de 8 personnages, 5 arbres et d'un décor, comme illustré en figure 4.36. Les propriétés spécifiées sur les objets concernent l'apparition à

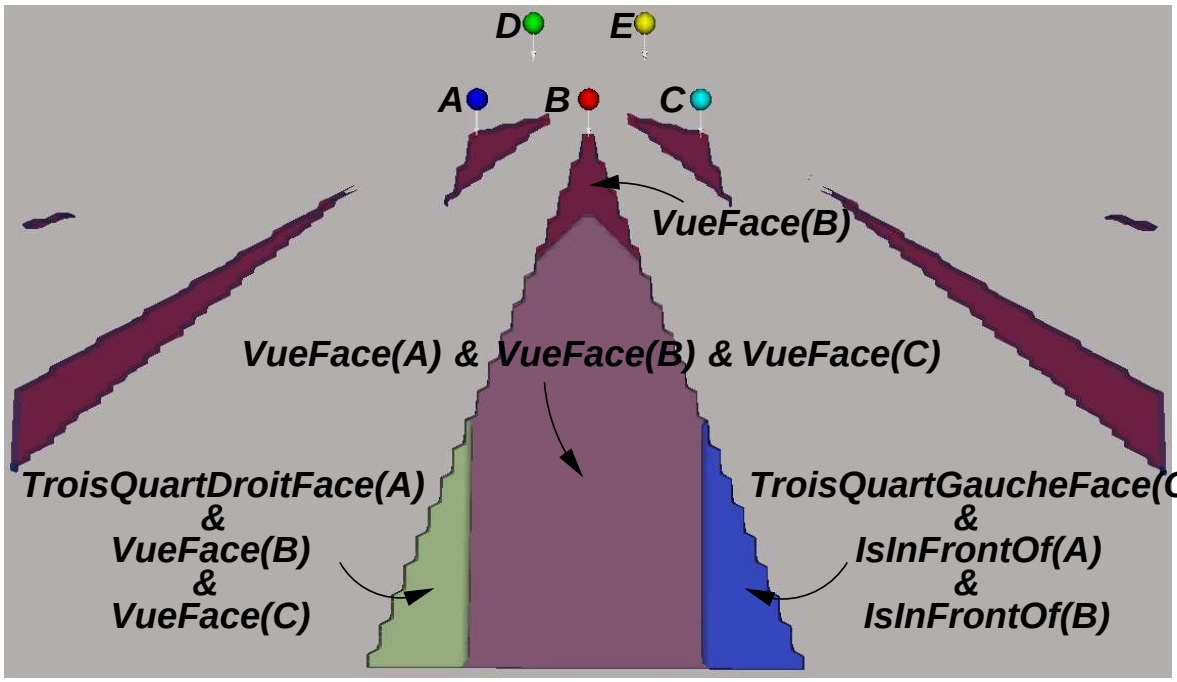



Figure 4.34 – Raffinements possibles d'une première description utilisateur.

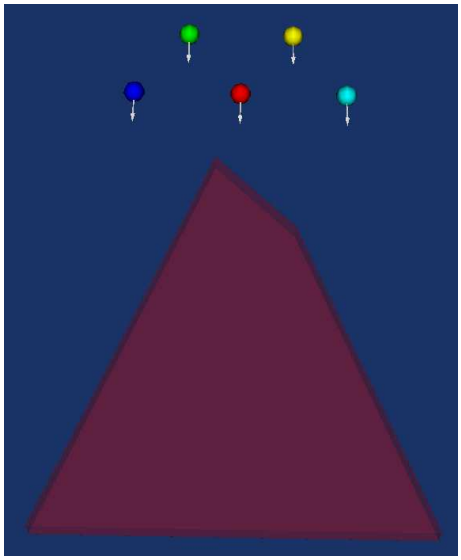


Figure 4.35 – Raffinement du problème des *five frames* pour lequel l'utilisateur a specifié sa volonté de voir les trois objets $A$, $B$ et $C$ de face.

l'écran des 8 personnages d'intérêt, ainsi que l'absence d'occlusions entre certains personnages et les arbres de la scène.

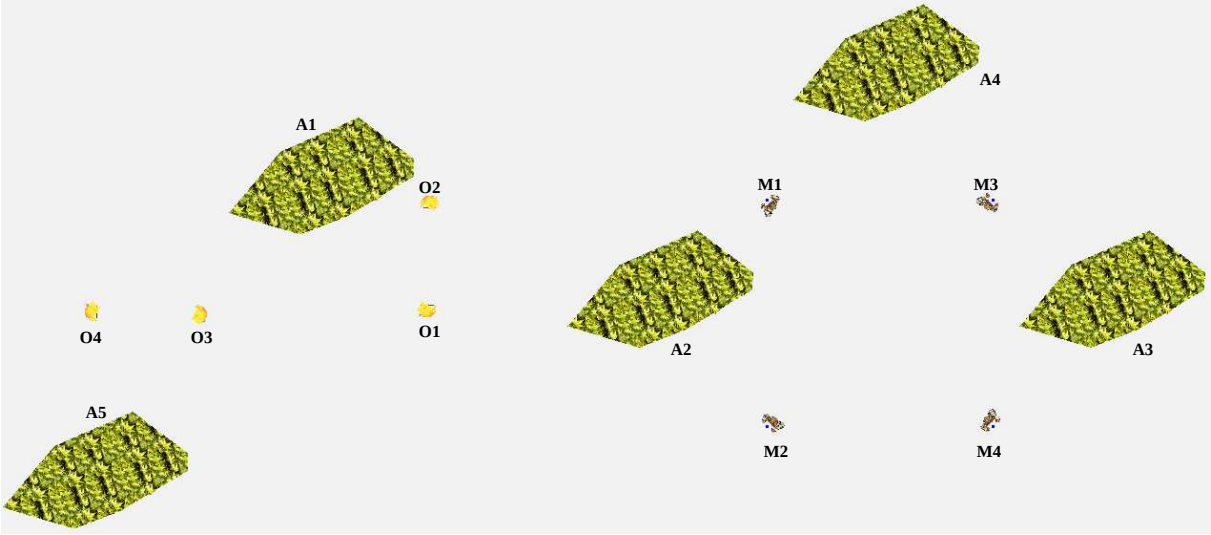



Figure 4.36 – Volume sémantique correspondant à la description du problème d'occlusion.

Les propriétés définies sur cette configuration 3D concernent le maintien à l'écran des quatre personnages $\{O1, O2, O3, O4\}$ et des quatre monstres $\{M1, M2, M3, M4\}$. L'utilisateur spécifie également des propriétés d'occlusions entre les personnages et les arbres de la scène. Ainsi, les quatre personnages ne doivent pas être occultés par les arbres, tout comme le monstre $M2$ par l'arbre $A1$. Le script suivant récapitule l'ensemble des propriétés définies par l'utilisateur :

```
#Occlusions pour l'objet $O1
Occlusion $O1 $A1 no
Occlusion $O1 $A2 no
Occlusion $O1 $A3 no
Occlusion $O1 $A4 no

#Occlusions pour l'objet $O2
Occlusion $O2 $A1 no
Occlusion $O2 $A2 no
Occlusion $O2 $A3 no
Occlusion $O2 $A4 no

#Occlusions pour l'objet $O3
Occlusion $O3 $A1 no
Occlusion $O3 $A2 no
Occlusion $O3 $A3 no
Occlusion $O3 $A4 no

#Occlusion pour le monstre $M2
Occlusion $M2 $A1 no
```

```
#Framing des quatre personnages
Frame $O1 -1.0 -1.0 1.0 1.0
Frame $O2 -1.0 -1.0 1.0 1.0
Frame $O3 -1.0 -1.0 1.0 1.0
Frame $O4 -1.0 -1.0 1.0 1.0
#Framing des quatre monstres
Frame $M1 -1.0 -1.0 1.0 1.0
Frame $M2 -1.0 -1.0 1.0 1.0
Frame $M3 -1.0 -1.0 1.0 1.0
Frame $M4 -1.0 -1.0 1.0 1.0

#Ajout d'un plan de coupe pour contraindre la position de la caméra
Plane 0 0 0 0 0 1
```

L'image 4.37 illustre le volume sémantique obtenu à l'issue du processus de filtrage géométrique de l'espace de recherche en fonction des propriétés décrites dans le script ci-dessus.

Figure 4.37 – Volume sémantique correspondant à la description du problème d'occlusion.

Enfin, la figure 4.38 propose une image résultat obtenue par application du processus numérique de recherche locale continue.

## 4.7 Conclusion

Les volumes sémantiques constituent une nouvelle approche d'aide au placement de caméra en environnements virtuels. En effet, contrairement aux approches existantes, la méthode des volumes sémantiques se focalise sur la manière d'apporter une aide à l'utilisateur plutôt que sur le fait de *simplement* trouver une solution à un problème donné.

L'approche permet de créer les différentes classes de solutions sémantiquement équivalentes à un problème de placement de caméra spécifié par un metteur en scène virtuel et propose à ce dernier de naviguer et de comparer les solutions obtenues. L'utilisateur peut ensuite raffiner la description initiale du problème, en caractérisant les volumes sémantiques résultats en fonction d'autres objets de la scène 3D ou par rapport à d'autres propriétés cinématographiques.

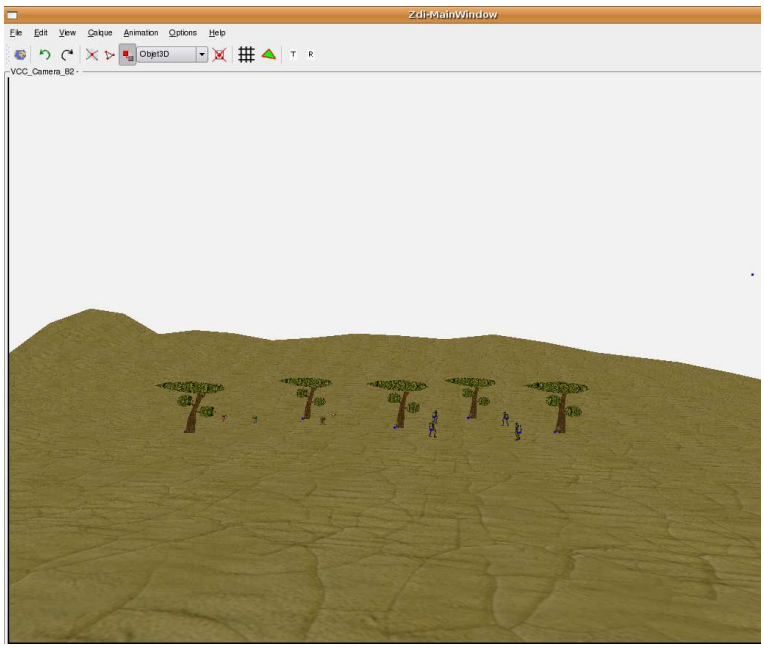



Figure 4.38 – Image résultat calculée par le processus numérique de résolution.

Les volumes sémantiques sont donc, tout comme l'algorithme MAX-NCSP présenté dans le chapitre 3, UNE solution permettant de répondre au problème de placement de caméra en environnements virtuels. Son originalité première réside dans le découpage géométrique de l'espace basé sur les idées de *BSP*, des aspects visuels et de *viewpoint space partitionning*, qui permet de traiter équitablement toutes les classes de solutions d'un problème de composition visuelle.

## 4.8 Discussion - Perspectives

Il apparaît clairement intéressant d'étendre l'approche des volumes sémantiques à des environnements dynamiques, afin de pouvoir calculer non plus des positions de caméra satisfaisant un ensemble de propriétés, mais des trajectoires respectant un ensemble de caractéristiques temporelles définies sur les objets d'une scène en mouvement.

La mise en œuvre d'une extension temporelle des volumes sémantiques, permettant de caractériser des trajectoires de caméra, nécessite un ensemble de propriétés ayant des dates de début ($t_d$) et de fin ($t_f$), définies sur les objets de la scène dont nous connaissons les trajectoires respectives. L'idée consiste à créer le volume sémantique associé à une propriété au temps $t_d$, puis en se basant sur la trajectoire de l'objet d'intérêt, à extruder le volume sémantique obtenu le long de cette trajectoire jusqu'au temps $t_f$. Le volume de balayage ainsi obtenu représente les positions de caméra cohérentes avec la propriété concernant l'objet d'intérêt entre les temps $t_d$ et $t_f$.

Lorsque la propriété est définie pour un couple d'objets ayant chacun des trajectoires différentes, nous faisons suivre au volume initial les deux trajectoires en parallèle et considérer l'union des deux volumes obtenus comme étant les zones acceptables pour les caméras.

L'implémentation de cette extension temporelle des volumes sémantiques nécessite un algorithme d'extrusion de volume le long d'une trajectoire, également appelé algorithme de balayage (*sweep algorithm*, pour plus d'informations se référer à [ww]). Or, si nous souhaitons représenter les volumes temporels en tant que surfaces implicites, il est nécessaire d'augmenter les définitions présentées dans la section 4.5.1 afin de prendre en compte une quatrième dimension : le *temps*. Une *surface implicite temporelle* est donc définie dans un espace à 4 dimensions, *i.e.* $F_t : \mathbb{R}^4 \rightarrow \mathbb{R}$ et permet de représenter l'évolution d'une surface implicite au cours du temps. L'intersection de ces surfaces implicites temporelles caractérise un volume 4D satisfaisant une conjonction de propriétés au cours du temps. La figure 4.39 illustre une intersection de volume sémantique au cours du temps, chaque volume étant projeté sur la composante $X$ de repère monde afin de pouvoir le représenter.

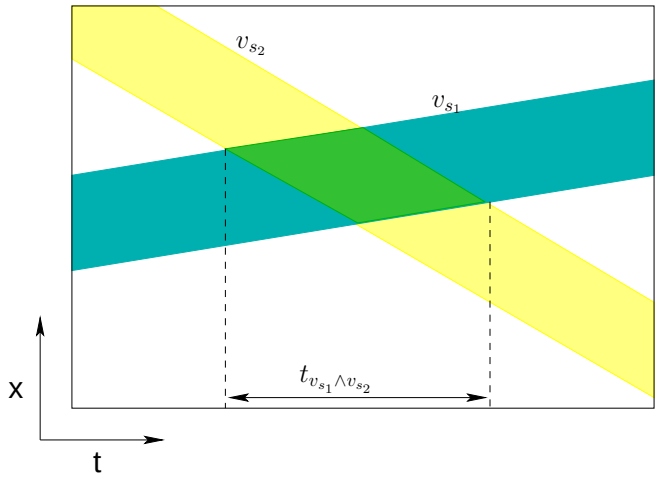



Figure 4.39 – Intersection volumes sémantiques temporels, la représentation des volumes est abstraite à la composante $X$ dans un souci de clarté de la représentation.

La zone diagonale verte située au centre de la figure correspond à l'intersection des volumes $v_{s_1}$ et $v_{s_2}$ pour l'intervalle de temps $t_{v_{s_1} \wedge v_{s_2}}$, ce qui signifie que la conjonction des propriétés $p_1$ et $p_2$ est vérifiée pour cet intervalle de temps. Comme pour la version statique des volumes sémantiques, nous devons tesseler les surfaces implicites afin d'obtenir les composantes connexes de l'intersection et donc les différentes classes de solution du problème.

Cependant, il n'existe pas à ce jour d'implémentation permettant d'approximer des surfaces implicites temporelles (4D) par des techniques de *Marching Cubes*. Il est donc impossible dans la pratique de mettre en œuvre les surfaces implicites temporelles telles que présentées ici.

Afin de pallier à cette limitation, nous avons étudié la pertinence d'une approximation de la dimension temporelle par l'union de volumes établit à des instants fixes (*e.g.* toutes les $x$ millisecondes). La figure 4.40 illustre l'idée de discrétisation de la dimensions temporelle appliquée à l'approche des volumes sémantiques.

La trajectoire est représentée par une flèche et le volume sémantique résultant de la rotation du volume $t_1$ pendant deux instants successifs ($t_2$ et $t_3$) est constitué de l'union Booléenne des trois volumes aux temps $t_1, t_2$ et $t_3$. Réaliser l'union de ces trois volumes caractérisant chacun trois instants différents aboutit à la formation d'un seul volume représentant la propriété pendant 3 subdivisions temporelles :

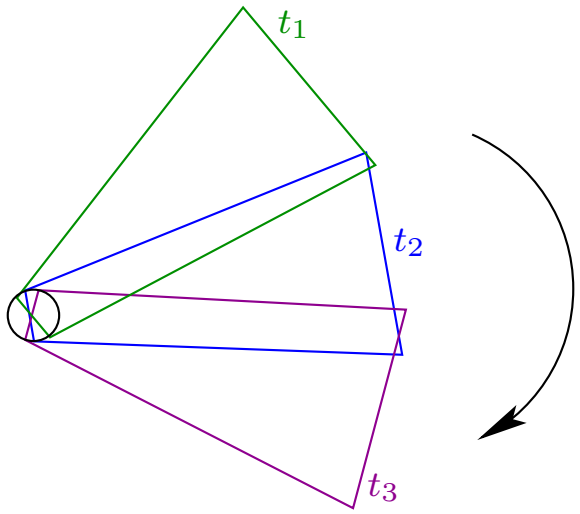



Figure 4.40 – Représentation d'un même volume sémantique à trois instants successifs $t_1, t_2$ et $t_3$.

$t_{1\to3}$.

Toutefois, cette représentation n'est pas satisfaisante puisqu'elle élude la notion temporelle de satisfaction de la propriété. En effet, l'approche peut fournir une configuration de caméra appartenant au volume $t_1$ comme étant solution d'une propriété à $t_2$ puisque la seule représentation que nous ayons est valable à la fois pour $t_1, t_2$ et $t_3$ sans distinctions ($t_{1\to3}$). Le résultat calculé est donc incohérent puisque le volume sémantique $t_1$ ne contient pas les positions de caméra consistantes au temps $t_2$.

Les deux solutions étudiées pour le mise en œuvre des volumes sémantiques temporels nous ont donc conduits dans des impasses, puisque la première est techniquement irréalisable et la seconde est incorrecte.

Afin de faciliter l'interaction avec l'utilisateur, il est possible de développer une interface graphique permettant de spécifier facilement et rapidement les propriétés cinématographiques sur les objets de la scène. Cette interface doit être pensée comme un greffon à rajouter dans les modeleurs 3D. Elle doit également permettre d'incorporer la notion de calques (ou *layers*) évoquée dans la section 4.2.5.2 lors de la description des propriétés de *framing*. Cet ajout autorise alors l'utilisateur à spécifier sans ambiguïtés les superpositions de cadres en mentionnant le niveau de profondeur associé à chacune des *frames* de l'image.

Enfin, la finalité visée de la méthode des volumes sémantiques concerne son intégration dans les modeleurs 3D actuels. Il est éventuellement possible de rajouter les étiquettes sémantiques associées à chacun des volumes lors de leur affichage, à la manière dont est annotée la capture d'écran 4.34. De cette manière, l'utilisateur peut choisir un volume bien précis en fonction de ses différentes étiquettes sémantiques, puis naviguer comme bon lui semble à l'intérieur de ce volume, afin de choisir la position de caméra qui lui convient le mieux.

# CHAPITRE 5
# La gestion de l'occlusion

« Vision is the art of seeing things invisible. »
Jonhatan Swift — *Thoughts on various subjects* tiré de *The Battle of the Books and Other Short Pieces*.

Indépendamment du domaine d'application visé, un des principaux challenges du contrôle automatique de caméra est de proposer aux utilisateurs des vues sans occlusions des objets d'intérêt d'une scène 3D. Par exemple, les jeux vidéo nécessitent de présenter le personnage principal et les objets (ou personnages) avec lesquels il interagit de façon à faciliter l'expérience du joueur, c'est-à-dire en évitant les occlusions. De même, les applications de visualisation scientifique ou de visualisation d'informations doivent favoriser la visibilité de propriétés et de relations lors de l'exploitation des données. Comme nous l'avons présenté dans les chapitres précédents (chapitres 2 et 4), une occlusion apparaît lorsque la projection d' un objet d'intérêt est recouverte à l'écran par celle d'un autre objet (ou d'un ensemble d'objets) de la scène. Bien que cette définition soit simple, la modélisation et la résolution d'une occlusion ne sont pas choses aisées. En effet, cette propriété très expressive peut être considérée de manière partielle (seulement une partie de la projection d'un objet est recouverte) ou totale (toute la projection est recouverte). Nous pouvons également nous intéresser à la gestion temporelle des occlusions (*i.e.* l'occlusion d'un objet au cours du temps) ou bien encore à des notions de reconnaissance d'un objet. Toutefois, bien que les occlusions sont en général préjudiciables à la transmission de l'information, elles permettent à certaines occasions de mieux percevoir les relations spatiales entre les objets de la scène, en particulier les distances des objets par rapport à la caméra (quel objet est devant quel autre) et peuvent se révéler profitables à la perception spatiale d'une scène 3D complexe par un utilisateur.

Nous avons insisté précédemment sur la relative pauvreté de la prise en compte de l'occlusion dans les approches existantes de contrôle de caméra. Nous pensons que cette relation est primordiale pour le contrôle de caméra. Ce problème est difficile, plus complexe que le seul évitement d'obstacles dans une scène et peut être assimilé à un problème de planification de trajectoire ayant sept degrés de liberté (les paramètres de la caméra). Un certain nombre de difficultés sont sous-jacentes à la gestion de l'occlusion en environnements virtuels et expliquent en partie la rareté de la littérature sur le sujet :

- comment planifier une trajectoire de caméra à 7 degrés de liberté en gérant les occlusions,
- comment déterminer de manière *précise* et *efficace* les situations d'occlusions dans des environnements généralement dynamiques et complexes,
- comment réagir à une situation d'occlusion, quelle direction choisir pour la caméra en fonction de la configuration de la scène et des risques potentiels d'occlusion,
- comment prendre en compte la dimension temporelle d'une occlusion, comment intégrer dans le processus de détection ou de réaction à l'occlusion les événements précédents de la scène, les prédictions sur les mouvements futurs ou bien encore tenir compte de la durée d'occlusion d'un objet à l'écran.

Conscients des difficultés inhérentes à la gestion de l'occlusion dans les environnements virtuels dynamiques et complexes, nous proposons dans ce chapitre une solution efficace et robuste de détection

et de réaction à l'occlusion. Notre méthode permet de prendre en compte une dimension temporelle en incorporant les évènements précédents dans la phase de choix des nouvelles positions de caméra.

## 5.1 La gestion de l'occlusion dans les méthodes existantes

Les approches de contrôle de caméra diffèrent grandement tant dans la façon de modéliser, que dans le degré de gestion de l'occlusion. En effet, certaines approches prennent en compte l'occlusion partielle (c'est le cas de l'approche des *volumes sémantiques* présentée au chapitre 4) ou permettent une quantification de l'occlusion d'un objet de la scène par un pourcentage (Bares *et al.* [BMBT00]) ou en nombre de pixels recouverts (Halper et Olivier [HO00]). Toutefois, dans ces deux exemples, l'occlusion partielle fait partie intégrante du processus de résolution mais pas du processus de modélisation. L'utilisateur est ainsi averti en cas d'occlusion partielle mais ne peut spécifier une contrainte de visibilité partielle d'un objet d'intérêt.

Dans les différents travaux, la prise en compte des occlusions partielles, même au sein du processus de résolution, reste anecdotique ; la majorité des approches ne considérant pas de différence entre l'occlusion totale et l'occlusion partielle. L'approche des *volumes sémantiques* reste la seule approche, à notre connaissance, permettant une distinction explicite dans la modélisation de l'occlusion partielle et de l'occlusion totale.

La relation d'occlusion est complexe, son expression dans les environnements dynamiques nécessite de pouvoir représenter les changements d'occlusion d'un objet au cours du temps, ceux-ci permettant d'inférer les relations spatiales entre les objets d'une scène. Prenons l'exemple d'un film dans lequel un personnage entre dans une voiture ou dans un bâtiment. Il est important d'assister le spectateur dans sa compréhension spatiale et temporelle des actions en faisant terminer un plan séquence par une vue occultée du personnage par la fermeture de la portière de la voiture ou de la porte d'entrée de l'immeuble et en commençant le plan suivant par une vue non occultée du même personnage à l'intérieur de sa voiture ou du bâtiment.

Les différences d'expressivité liées à la notion d'occlusion ont un impact significatif sur la modélisation et la résolution du problème. Nous pouvons classer les approches de gestion d'occlusion dans des environnements dynamiques en deux catégories, en fonction de la connaissance *a priori* qu'elles ont de la scène et du niveau de non déterminisme de l'environnement. Nous pouvons les diviser entre les approches réactives (qui ne possèdent aucune connaissance de la scène) et les approches « omniscientes » (qui ont une connaissance totale de la géométrie et des mouvements de la scène).

### 5.1.1 Techniques réactives de gestion de l'occlusion

Les techniques réactives n'ont aucune connaissance *a priori* des actions de l'environnement. Celles-ci sont utilisées dans les environnements fortement dynamiques, comme les jeux vidéo, les applications de suivi de cibles ou bien encore d'apprentissage interactif, elles se doivent donc d'être très performantes.

Une solution simple et performante repose sur la méthode de lancer de rayons (*ray-casting*, cf. [FvDFH90, AMH02]) depuis la caméra vers (*i*) l'objet d'intérêt pour détecter les situations d'occlusion ou (*ii*) dans des directions arbitraires de l'espace, afin de prévoir les occlusions avec le décor et/ou les autres objets de la scène, et déplacer la caméra en conséquence. Dans le système de gestion de caméra (*camera manager*) intégré au jeu *Full Spectrum Warrior* (*FSW*), John Giors [Gio04] implémente une méthode innovante de *ray-casting*, illustrée en figure 5.1. *FSW* est un jeu de combat tactique dans lequel le joueur doit gérer une escouade de combattants. De par cette spécificité (la gestion d'une équipe

et non d'un seul joueur), les systèmes de caméra conventionnels pour les jeux à la troisième personne (*TPS*, cf. section 2.1.2) ne sont pas satisfaisants. Le système *Autolook* mis en place dans *FSW* permet de modifier uniquement l'orientation de la caméra en fonction des résultats d'intersections des rayons lancés vers la gauche et la droite de la caméra (cf. figure 5.1). La modification du vecteur vision de la caméra permet d'éviter que des éléments du décor ne recouvrent une trop grande partie de l'image finale et assure ainsi une visibilité maximale des membres de l'équipe. Les collisions de *FSW* sont prises en compte de manière similaire. La prévision et l'évitement de collisions sont gérées en lançant des rayons depuis l'objet d'intérêt vers le voisinage proche de la caméra et en étudiant les résultats des intersections avec le décor.

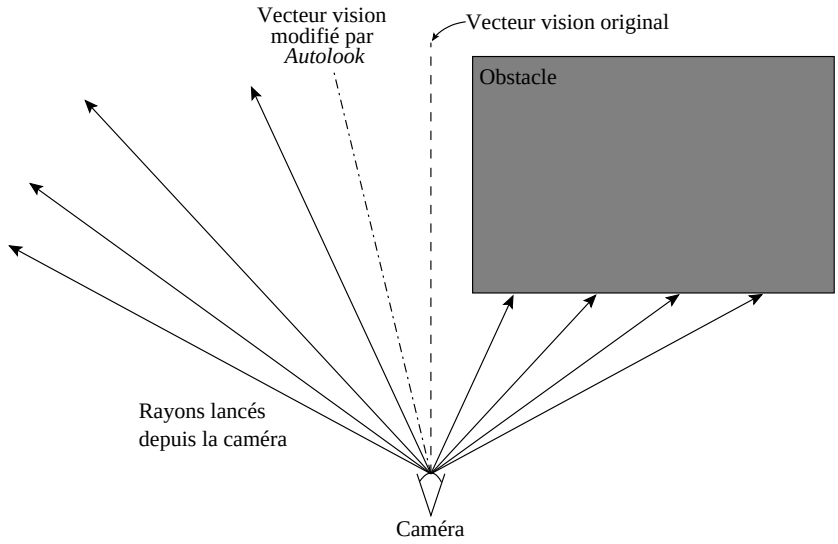



Figure 5.1 – Illustration du système *Autolook* [Gio04] en vue de dessus. Le système lance des rayons à droite et à gauche de la position courante de caméra. En fonction des intersections, le vecteur vision est modifié afin de limiter la place des obstacles à l'écran. Ici les rayons gauches ne rencontrent aucune intersection alors que ceux de droite rencontrent un obstacle. Le vecteur vision est donc décalé sur la gauche afin de limiter l'affichage de l'obstacle à l'écran (*Autolook* ne modifie que l'orientation de la caméra, pas sa position).

Le *ray-casting* a également été appliqué à des environnements d'apprentissage interactifs, dans lesquels les étudiants contrôlent un avatar virtuel et explorent l'environnement à la recherche d'informations. Bares *et al.* [BRZL98] ont recours à cette technique afin d'assurer des prises de vues non occultées de l'avatar et des éléments importants de l'environnement. Si des occlusions sont détectées, la caméra est alors déplacée et tourne progressivement autour de l'avatar jusqu'à obtention d'une vue non obstruée.

Dans leur système de gestion de composition visuelle 3D, Bares *et al.* [BMBT00] utilisent une technique de lancer de rayons légèrement différente : les personnages sont représentés *via* leurs boîtes englobantes et l'occlusion d'un objet est testée en lançant neuf rayons depuis la caméra vers les huit sommets et le centre de sa boîte englobante. Le nombre d'intersections détecté pour chacun de ces neuf rayons permet alors d'estimer un pourcentage d'occlusion de l'objet concerné. Cette estimation permet au système d'évaluer la qualité des prises de vue et de favoriser celles qui minimisent les occlusions partielles.

Enfin, dans leur système de cinématographie automatique permettant de véhiculer des émotions dans un environnement d'agents autonomes, Tomlinson *et al.* [TBN00] ont également recours à des techniques de *ray-casting*. Une fois que le système a satisfait toutes les contraintes de positionnement des éléments pour la prise de vue courante, des rayons sont lancés depuis la caméra vers l'objet d'intérêt afin de vérifier la visibilité de ce dernier. En cas d'échec, la caméra est déplacée en fonction des intersections réalisées.

L'efficacité des techniques de *ray-casting*, alliée à leur simplicité de mise en œuvre les rend particulièrement intéressantes pour les applications temps réel. Les performances des ces méthodes peuvent être améliorées en remplaçant les calculs d'intersections rayon-objet par des tests rayon-volume englobant de l'objet. Le choix de l'englobant (sphère ou boîte englobante, arbre de boîtes englobantes orientées, hiérarchie de sphères englobantes, etc.), ainsi que le nombre de rayons lancés, déterminent la précision et le coût des tests d'occlusion réalisés. Bien que le *ray-casting* consiste en une méthode simple et performante de prise en compte de l'occlusion, la détection sous-jacente est toutefois largement partielle de par la nature des tests effectués (les rayons doivent être suffisamment nombreux pour intersecter les éléments de la scène).

La prise en compte de l'occlusion en environnements dynamiques peut également être effectuée par construction de *régions consistantes* de l'espace pour lesquelles aucune occlusion ne peut avoir lieu. La première approche ayant proposé la construction de telles régions est celle de Phillips *et al.* [PBG92] qui l'ont incorporé dans le système JACK. L'idée est empruntée à une amélioration d'une technique de calcul d'illumination globale (*i.e.* de radiosité) dans une scène 3D, à savoir la méthode de l'« hémicube », proposée initialement par Cohen et Greenberg [CG85]. Les auteurs proposent de centrer un cube sur l'objet d'intérêt et de l'orienter vers la position de caméra courante afin d'obtenir une carte de visibilité de tout l'environnement par projection des objets sur l'hémicube. Si la caméra est obstruée, il faut chercher dans le voisinage de son orientation courante une zone vierge dans l'hémicube qui caractérise un bon candidat pour l'obtention d'une vue non occultée de l'objet d'intérêt. Il suffit ensuite de calculer les angles de rotation permettant de modifier le vecteur *look-at* courant de la caméra pour obtenir celui correspondant à la zone libre de l'hémicube.

L'idée de construction de régions consistantes basées sur une projection de l'environnement vers l'objet d'intérêt a été reprise à l'identique par Drucker et Zeltzer dans leur système CAMDROID [DZ95]. Bares et Lester [BL99] se sont également basés sur ces travaux, mais proposent de projeter toutes les boîtes englobantes des objets de l'environnement sur une sphère centrée sur l'objet d'intérêt $O$. Les projections des objets de la scène sur cette sphère sont ensuite converties dans un système de coordonnées sphérique global, puis inversées afin de représenter les régions de l'espace sans occlusions pour la visualisation de l'objet $O$.

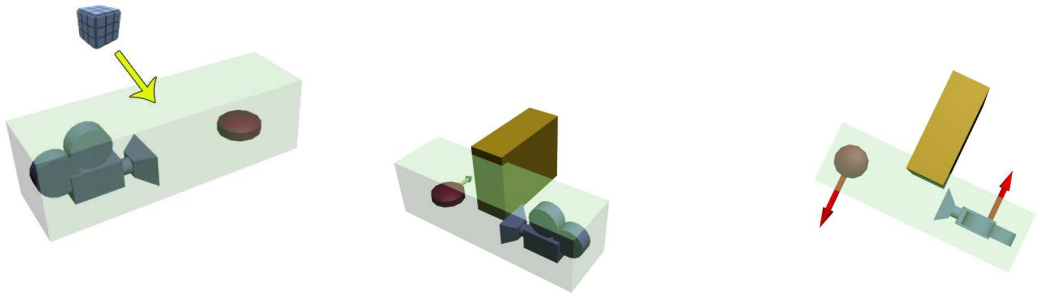
Toujours dans les environnements fortement dynamiques, les techniques d'asservissement visuel (cf. section 2.4) assurent un évitement des occlusions par un contrôle basé image de la caméra. Les techniques réactives basées image nécessitent la définition de fonctions permettant d'assurer la satisfaction d'un certain nombre de tâches visuelles. La prise en compte de l'occlusion requiert alors la définition d'une fonction atteignant une valeur infinie lorsque l'objet d'intérêt est occulté à l'écran. Cette dernière a été proposée par Marchand et Hager [MH98], puis réutilisée par Marchand et Courty [MC00, CM01, MC02, Cou02]. Soit $I$ l'objet d'intérêt de l'application référencée vision dont la tâche principale consiste à conserver la projection de $I$ à une position donnée à l'écran. Soit $O = \{O_1, \ldots, O_n\}$ la projection d'un ensemble d'objets potentiellement occultants, la fonction d'évitement d'occlusion en tant que tâche secondaire en asservissement visuel est définie comme:

$$f = \frac{1}{2}\alpha \sum_{i=1}^{n} e^{-\beta(\|\mathsf{proj}(I) - \mathsf{proj}(O_i)\|^2)} \tag{5.1}$$

où $\mathsf{proj}(X)$ représente la projection de l'objet $X$ à l'écran. $\alpha$ et $\beta$ sont deux scalaires caractérisant respectivement l'amplitude de la réaction de la caméra (plus $\alpha$ est grand, plus la vitesse de la caméra est importante) et le moment à partir duquel l'évitement de l'occultation est activé (plus $\beta$ est grand, plus la projection de l'objet occultant peut se rapprocher de $\mathsf{proj}(I)$ avant que la réaction de la caméra ne se déclenche). La tâche secondaire est ensuite exprimée en fonction de cette fonction $f$ et des matrices d'interaction de la caméra (cf. section 2.4).

Courty et Marchand [CM01, MC02, Cou02] proposent également d'avoir recours à l'utilisation de volumes englobants afin de gérer les situations d'occlusion. Les auteurs isolent différents cas de figure pouvant amener à l'occlusion d'un objet d'intérêt à l'écran (cf. figure 5.2) :

- mouvement d'un objet de la scène vers l'objet d'intérêt,
- déplacement de l'objet d'intérêt lui-même,
- déplacement de la caméra.



(a) Occlusion due au déplacement d'un objet de la scène. (b) Occlusion due au déplacement de l'objet d'intérêt. (c) Occlusion due au déplacement de la caméra.

Figure 5.2 – Évitement d'occlusions par définition d'un volume englobant la caméra et l'objet d'intérêt, d'après [Cou02].

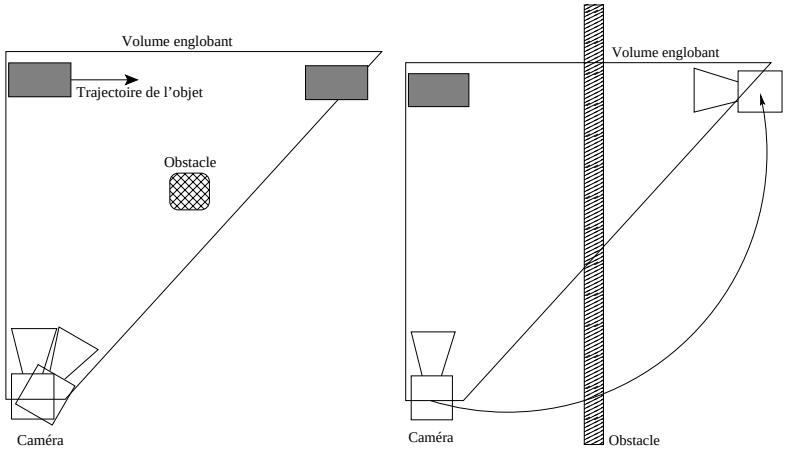
Afin d'anticiper les occlusions dans une scène dynamique, Courty et Marchand proposent de créer des volumes qui englobent à la fois l'objet d'intérêt et la caméra, incorporant ainsi un caractère prédictif à la méthode. La création de ces volumes prédictifs, permettant de gérer à la fois une occlusion et une collision future, est illustrée en figure 5.3. Lorsqu'une intersection entre le décor et les volumes prédictifs est détectée, il y a soit :

- occlusion de l'objet d'intérêt si celui-ci se déplace (cf. figure 5.3 (a)),
- collision entre la caméra et le décor lors d'un déplacement de celle-ci (cf. figure 5.3(b)).

Ces prédictions sont toutefois raisonnablement valides si la trajectoire de l'objet d'intérêt reste cohérente avec le mouvement approximé.

Il reste cependant des cas pathologiques qui nécessitent un traitement spécifique, par exemple lorsque la caméra doit suivre un objet dans un couloir présentant des virages à angles droits. Dans ce cas, les mouvements de caméra sont très limités (impossibilité de sortir du couloir ou de traverser les murs) et l'approche à adopter lors d'un virage de l'objet consiste à réduire la distance entre la caméra et l'objet jusqu'à ce que l'occlusion ou la collision disparaisse. Cet exemple particulièrement difficile est souvent

rencontré dans les jeux vidéo, par exemple lorsque le personnage joueur traverse un couloir lors de l'exploration d'un bâtiment. Les systèmes classiques de gestion de caméra tendent dans ce cas à modifier les valeurs de transparence des objets de la scène en permettant à la caméra de se déplacer librement à travers le décor. Les personnages restent alors visibles et la perception spatiale du joueur par rapport à l'environnement est conservée. Toutefois, cette approche souffre des limitations des approches basées image, présentées dans la section 2.4.




(a) Détection d'une future occlusion. (b) Détection d'une future collision avec le décor.

Figure 5.3 – Prévision d'occlusion et/ou de collision grâce à des volumes prédictifs, d'après [Cou02].

Halper *et al.* [HHS01] ont proposé un système de gestion de caméra spécialement conçu pour les environnements de jeux vidéo. Le principe d'évitement d'occlusions au sein de ce système consiste en la construction de régions de visibilité potentielle (*Potential Visibility Regions – PVR*). Dans cette approche, le *game designer* (personne chargée de construire les niveaux d'un jeu) ou le développeur en charge de la gestion de la caméra, peuvent imposer des contraintes sur les mouvements de cette dernière afin d'éviter les occlusions. Ces contraintes sont définies par un ensemble de polygones représentant des contraintes géométriques de préférence, spécifiées de la manière suivante : les régions favorisées sont modélisées de façon plus claire (lumineuse) que les régions de moindre préférence. Ces polygones représentent les *PVR* vers lesquels la caméra doit préférentiellement se déplacer afin d'obtenir des vues non obstruées des objets d'intérêt.

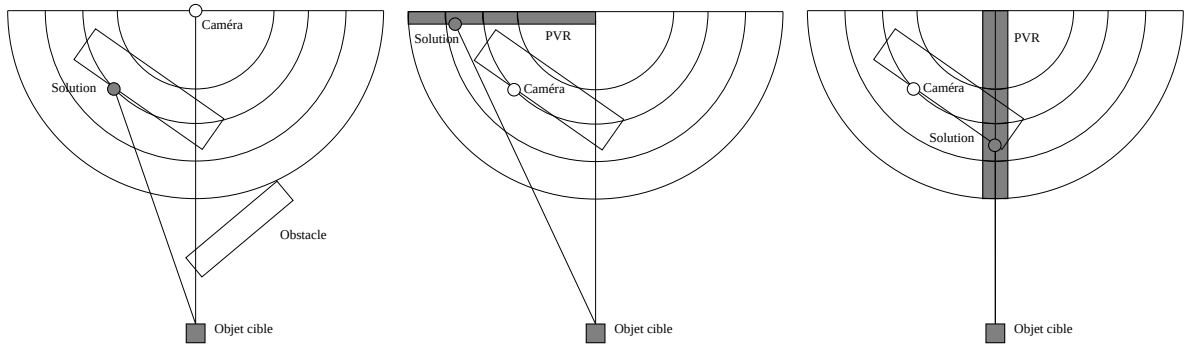
L'idée consiste à « écrire » dans une image seulement les informations de profondeur liées aux objets potentiellement occultants, puis de rendre les *PVR* dans le même *buffer* en affichant les régions des plus préférentielles (les plus claires) à celles de moindre intérêt (les plus sombres). Les régions les plus claires sont écrites dans le *stencil buffer* afin de ne pas être recouvertes par les régions plus foncées. Le résultat est une image dans laquelle les couleurs les plus lumineuses (ayant passées en premier le test de profondeur) sont visibles (grâce au *stencil buffer*). La couleur la plus lumineuse de l'image représente donc la position la plus intéressante, celle vers laquelle la caméra doit se déplacer.

Cette méthode permet une certaine flexibilité de par la spécification des *PVR*. La figure 5.4 illustre comment un *game designer* peut contraindre les mouvements de la caméra en définissant des régions de visibilité potentielles. De plus, la définition de ces *PVR* permet d'assurer la satisfaction d'autres

contraintes visuelles prises en compte dans le système de gestion de caméra de Halper *et al.* En effet, si l'utilisateur souhaite assurer une contrainte d'orientation entre la caméra et l'objet d'intérêt, un *PVR* peut être défini dans ce sens. Par exemple, la figure 5.4 (c) illustre la création d'un *PVR* qui assure de voir l'objet d'intérêt de face, en favorisant les régions dans l'alignement de l'objet cible.

La figure 5.4 illustre le concept de *PVR* grâce à un exemple, la figure 5.4 (a) correspondant à la recherche basique d'un bon point de vue. Les cercles concentriques ont une intensité lumineuse décroissante et représentent les zones de recherche à favoriser (du plus proche de la caméra : le plus clair ; au plus éloigné : le plus sombre) de part et d'autre de la ligne de vue désirée (la droite verticale). La vue de la caméra étant obstruée par des éléments de la scène, la recherche d'un bon point de vue est nécessaire. Celle-ci s'effectue le long des cercles concentriques en partant du plus proche (ayant la préférence maximale car étant le plus lumineux) vers le plus éloigné. La solution est trouvée le long du troisième cercle, au premier point permettant de voir la cible de façon non occultée.

Les schémas 5.4 (b) et (c) illustrent la définition de deux *PVR* (les rectangles gris) permettant de contraindre les déplacements de la caméra en fonction de propriétés définies par le *game designer*. La recherche de bon point de vue s'effectue toujours le long des cercles concentriques, mais est favorisée dans les *PVR*. Ainsi, dans le schéma central (figure 5.4 (b)), l'utilisateur souhaite privilégier les positions de caméra amenant à une image de profil gauche de l'objet cible, alors que le *PVR* de l'image 5.4 (c) va favoriser une vue de face de l'objet d'intérêt.




(a) Recherche non contrainte d'un bon point de vue grâce à la méthode *PVR*.

(b) Recherche sous contrainte : l'utilisateur a spécifié un *PVR* favorisant l'obtention d'un profil gauche de l'objet ciblé.

(c) Recherche sous contrainte : l'utilisateur a spécifié un *PVR* favorisant l'obtention d'une vue de face de l'objet d'intérêt

Figure 5.4 – Illustration de recherche d'un bon point de vue grâce au concept de *PVR (Potential Visibility Region)* proposé par Halper *et al.* [HHS01].

L'implémentation de cette méthode est effectuée *via* un algorithme projectif efficace de calcul de volumes d'ombrage (*shadow volumes*). Les *buffers* utilisés lors des rendus colorimétriques peuvent être de taille relativement restreinte (de l'ordre de $32 \times 32$ pixels) sans perte de précision préjudiciable à l'évitement de l'occlusion. Enfin, afin d'améliorer les performances de l'algorithme, les auteurs proposent d'abstraire les éléments du décor et autres objets potentiellement occultants lors du rendu colorimétrique par des englobants primitifs plus rapidement affichables. L'avantage des rendus matériels, outre leur efficacité, réside dans la précision obtenue lors du rendu des objets si l'on n'a pas recours à des englobants. Les occlusions sont alors détectées de manière beaucoup plus précise que par les autres méthodes présentées. Cette technique souffre toutefois de la nécessité d'interaction avec le *game designer* pour la

définition des *PVR* et de sa relative difficulté de mise en œuvre.

### 5.1.2 Approches « omniscientes » de gestion de l'occlusion

Au contraire des approches réactives, les approches « omniscientes » ont une connaissance totale de l'environnement dans lequel évolue la caméra. Nous retrouvons toutefois des méthodes analogues pour la gestion de l'occlusion, à savoir la création de régions consistantes ou l'utilisation de rendus matériels.

Concernant la création de régions consistantes, Pickering [Pic02] propose une implémentation basée sur une adaptation de l'algorithme de calcul des volumes d'ombrage (*shadow volumes*), présenté en figure 5.5. L'idée est de considérer le centre de l'objet d'intérêt comme étant l'œil et de calculer ensuite chacun des *shadow volumes* correspondant aux éléments potentiellement occultants de la scène en considérant leur centre comme une source lumineuse.

Les régions de l'espace de recherche n'appartenant à aucun volume d'ombrage correspondent aux positions de caméra permettant d'obtenir une image non occultée de l'objet d'intérêt. Ces régions potentielles sont obtenues en intersectant l'espace de recherche initial à chacun des *shadow volumes*. Pour des raisons d'efficacité, le calcul des volumes d'ombrage est effectué en abstrayant les occultants potentiels par leurs boîtes englobantes.

L'approche des volumes sémantiques (cf. chapitre 4) est elle aussi basée sur une abstraction des objets de la scène par une sphère englobante. Cette méthode peut être définie pour tout couple d'objets de la scène et permet une caractérisation des régions de l'espace en fonction de l'occlusion partielle, totale ou de l'absence d'occlusion entre ces deux objets. Une caractérisation complète de la scène est possible en combinant les propriétés d'occlusions pour tous les objets de la scène et permet ainsi d'assurer à un objet d'apparaître complètement (non-occlusion), partiellement (occlusion partielle) ou pas du tout (occlusion totale) dans l'image résultat.

Enfin, Halper et Olivier dans leur système CAMPLAN [HO00] proposent une prise en compte de l'occlusion en plusieurs étapes, mélangeant les méthodes de caractérisation de l'espace et de rendu matériel. Dans CAMPLAN, les éléments de la scène sont représentés sous forme de hiérarchie de sphères englobantes (cf. figure 5.6). La méthode se décompose en trois étapes :

1. la détermination du nombre d'objets potentiellement occultants. Ceci est réalisé en testant les recouvrements des hiérarchies de sphères englobantes du ou des objet(s) d'intérêt avec celles des objets potentiellement occultants. Si cette évaluation ne conduit pas à la détection d'occlusions, l'objet d'intérêt ne peut être occulté.
2. l'effacement d'un tampon image (*frame buffer*) puis le rendu colorimétrique (*i.e.* avec une couleur par facette et sans source lumineuse) de tous les objets à évaluer dans ce tampon image. Le *frame buffer* est ensuite lu et chaque pixel ayant une valeur différente de la couleur de fond utilisée lors du rendu (généralement 0) est comptabilisé. Le nombre $P$ de pixels ainsi obtenu correspond à la projection des objets à évaluer (*i.e.* des objets d'intérêt).
3. le rendu de tous les objets potentiellement occultant dans ce même tampon image avec la couleur correspondant à la couleur d'effacement des tampons image. Les objets potentiellement occultants obtenus lors de la première étape sont affichés avec le test de profondeur activé. En effet, la détermination des surfaces visibles est nécessaire seulement entre les occultants et les objets d'intérêt. Il faut ensuite déterminer le nombre $V$ de pixels visibles du *frame buffer*, c'est-à-dire ayant une valeur supérieure à 0. L'état d'occlusion des objets d'intérêt consiste ainsi au rapport entre le nombre de pixels des objets d'intérêt projetés $P$ et celui des pixels visibles $V$ obtenu à l'issue de cette étape, *i.e.* $V/P$.

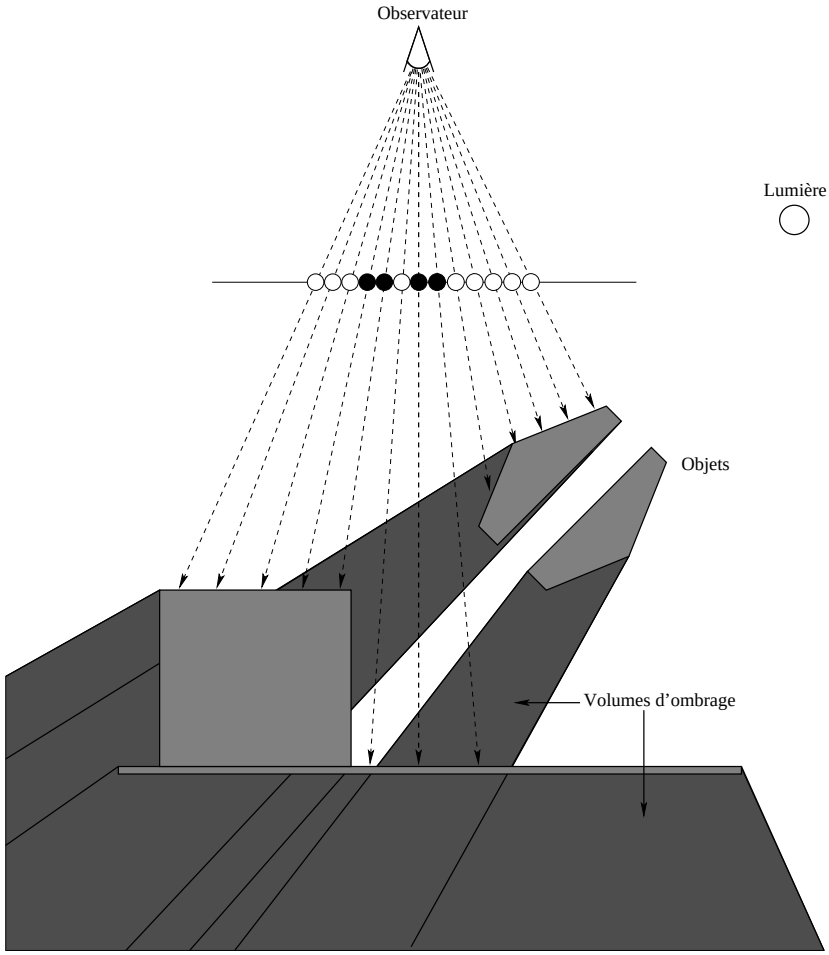
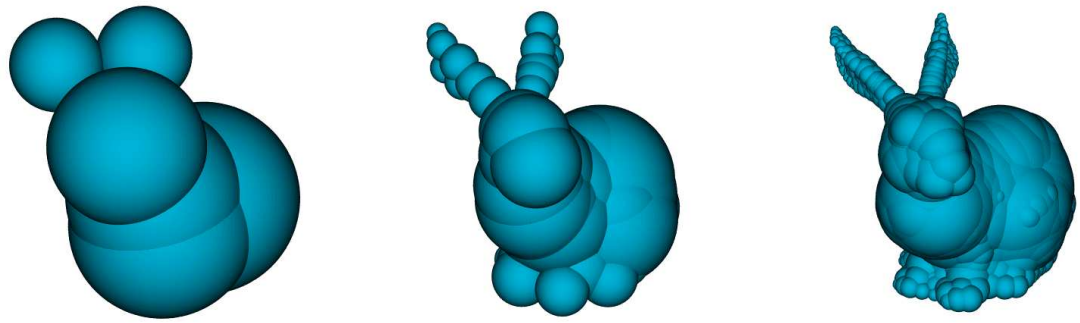



Figure 5.5 – Algorithme de calcul des volumes d'ombrage. La source lumineuse permet de définir les zones d'ombrage en fonction de la position des objets dans la scène. Afin de déterminer si un pixel est à l'ombre, il faut tester l'appartenance à un volume d'ombrage de l'intersection entre le rayon correspondant à ce pixel et la scène 3D. Si l'intersection appartient à un *shadow volume*, le point est à l'ombre, sinon il est éclairé.



(a) Création de l'arbre de sphères englobantes à un seul niveau de subdivision. (b) Arbre de sphères englobantes à deux niveaux de subdivisions. (c) Arbre de sphères englobantes ayant trois niveaux de subdivisions.

Figure 5.6 – Création d'un arbre de sphères englobantes pour un modèle de lapin, illustré à différents niveaux de subdivisions, d'après Bradshaw et O'Sullivan [BO02].

L'évaluation de la différence de précision lors de l'utilisation de tampons image ayant des résolutions allant de $32 \times 32$ à $640 \times 480$ pixels s'est montrée dans la plupart des cas insignifiante. Les auteurs préconisent donc l'utilisation de tampons réduits afin d'accélérer le processus. Il est à noter que des techniques innovantes de détermination de surfaces visibles (en particulier l'utilisation des extensions *occlusion queries* d'OpenGL) est possible pour améliorer les performances. Toutefois, les auteurs ont remarqué que le goulet d'étranglement de l'approche se situe au niveau du transfert des *buffers* de la mémoire de la carte graphique (GPU) vers la mémoire du processeur (CPU). Ainsi la méthode présentée ici suffit à obtenir des performances acceptables en fonction du nombre de polygones des scènes étudiées.

### 5.1.3 Conclusion sur les méthodes existantes

Les approches présentées ici, qu'elles soient réactives ou non, souffrent de limitations dans leur prise en compte de l'occlusion dans une scène 3D, en effet soit elles :

- ne permettent pas de différencier les occlusions totales des occlusions partielles,
- ne peuvent assurer la prise en compte d'autres propriétés en sus de l'évitement de l'occlusion,
- ne sont pas adaptables aux environnements dynamiques,
- ne sont pas généralisables à plusieurs objets d'intérêt.

Soucieux de proposer une solution adaptable aux environnements dynamiques, ainsi que de prendre en compte des notions cinématographiques de haut niveau, nous proposons une approche de gestion d'occlusion d'un ensemble d'objets d'intérêt à l'écran dans un environnement 3D temps réel dynamique réactif.

## 5.2 Une nouvelle approche de gestion de l'occlusion : l'OCCLUSION AVOIDER

Notre approche vise à prendre en compte l'occlusion de deux objets à l'écran de manière robuste et efficace est implémentée au sein d'un prototype : l'OCCLUSION AVOIDER. La gestion simultanée de deux objets d'intérêt au sein d'une méthode d'évitement d'occlusion en environnement virtuel complexe et dynamique, présente un certain nombre d'avantages. En effet, dans le cadre d'applications interactives de gestion d'avatars virtuels, l'évitement d'occlusion de deux objets permet d'assurer leur visibilité de façon simultanée à l'écran, en particulier lors d'un dialogue entre deux avatars. De même, dans les jeux vidéo, la garantie de non occlusion de deux objets à l'écran permet de maintenir visibles à la fois le personnage principal et un objectif important du jeu (ennemi à combattre, endroit à atteindre, objet à récupérer, etc.).

Notre idée est illustrée par la figure 5.7 et est basée sur l'utilisation :

- de rendus matériel dans des tampons de profondeur depuis les objets d'intérêt vers la caméra (*backprojection*),
- d'une méthode de lancer de rayons (*ray-casting*) pour la construction de points potentiellement intéressants pour les prochaines positions de caméra.

Notre approche se distingue toutefois des techniques classiques de *ray-casting* puisque nous avons choisi de ne pas détecter les occlusions en testant des intersections entre des rayons issus de la caméra et les éléments de la scène 3D, mais plutôt de profiter des capacités du matériel graphique actuel et d'effectuer des rendus de la scène dans des tampons image (*frame buffer*). Notre approche se résume en cinq étapes :

1. effectuer un rendu de la scène dans des tampons de profondeur (*depth buffers*) depuis chacun des objets d'intérêt vers un écran situé à proximité de la caméra. Les points d'intersection sont situés le long des « rayons » issus des objets et dirigés vers les pixels de l'écran.

2. générer des rayons à partir des objets d'intérêt vers un écran virtuel situé dans le voisinage de la caméra. Les rayons sont construits de manière à s'intersecter en une grille de points 3D : les candidats aux futures positions de caméra.

3. effectuer les rendus objets dans des tampons de profondeur, puis étudier les *depth buffers* afin de déterminer si les points d'intersection correspondent à des positions occultées ou non. En effet, si le pixel correspondant au rayon issu de l'objet est « vide », cela signifie qu'aucun occultant n'est situé entre l'objet et la caméra. Tous les points situés sur ce rayon sont ainsi potentiellement solutions. Au contraire, si le pixel contient une information de profondeur, nous devons calculer la position 3D dans la scène de l'occultant, en fonction de la valeur contenue dans le *depth buffer*. Ceci nous permet de déterminer si les points d'intersection doivent être considérés comme solutions potentielles ou non, cf. figure 5.7.

4. stocker, pour chacun des points d'intersection, l'information d'occlusion par rapport aux objets pour plusieurs images successives afin de prendre en compte une dimension temporelle. Ces données sont accumulées puis combinées afin d'établir le statut courant des intersections candidates.

5. choisir le meilleur point d'intersection pour la prochaine position de caméra en fonction des occlusions précédentes (cf. étape précédente) et d'une heuristique. Celle-ci peut concerner la cohérence à l'écran, *i.e.* vouloir minimiser le déplacement entre deux positions successives de caméra, assurer une distance constante par rapport à un des objets d'intérêt, etc.

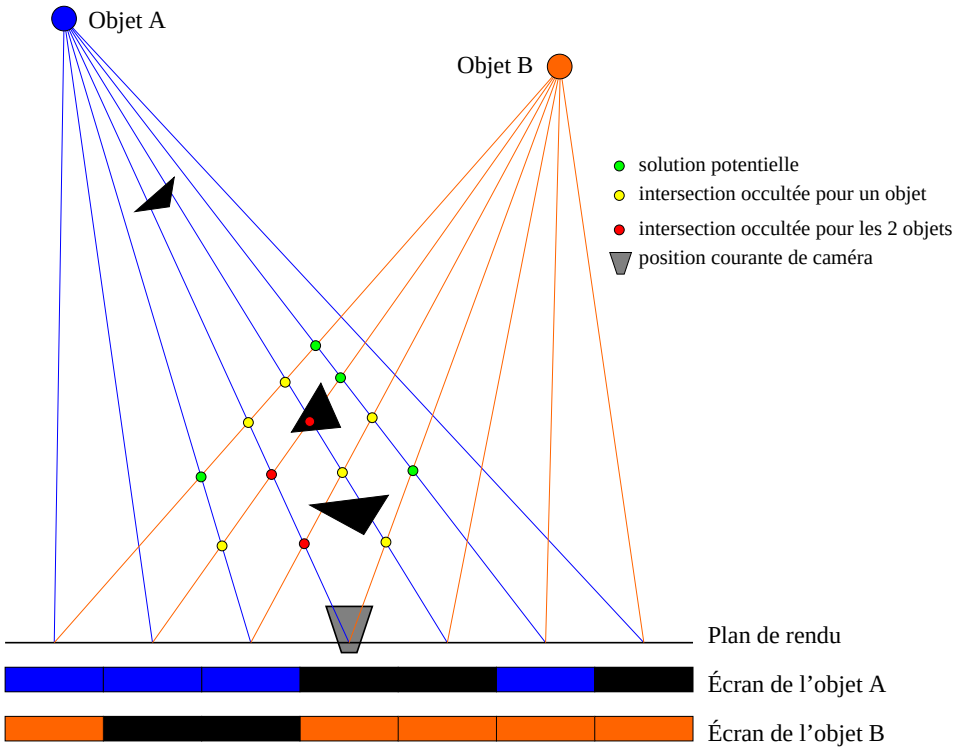



Figure 5.7 – Illustration de notre approche d'évitement de l'occlusion. Des rendus de la scène sont effectués depuis les objets dans des tampons de profondeur (les écrans) afin de déterminer si les points d'intersection correspondent à des positions non occultées de caméra (points vert), occultées pour un des objets (points jaune) ou pour les deux objets d'intérêt (points rouge). Les occlusions sont détectées grâce aux distances entre les points candidats, les objets occultants (triangles noirs) et les objets d'intérêt.

La deuxième étape de l'algorithme nécessite la définition de pyramides de vue permettant d'effectuer les rendus objets à partir des positions des objets d'intérêt et de la caméra. Les informations obtenues par l'étude des tampons de profondeur nous permettent d'inférer les positions de caméra pour lesquelles les objets d'intérêt ne sont pas occultés. Toutefois, nous devons reproduire les paramètres de la caméra pour que ces rendus objets soient exploitables. Enfin, les scènes étudiées étant dynamiques, il nous faut intégrer un mécanisme permettant la prise en compte des mouvements de caméra.

La figure 5.8 présente une capture d'écran de l'OCCLUSION AVOIDER où sont affichés la sphère des positions futures de caméra (représentée par une sphère jaune), le plan de rendu (représenté par un rectangle rouge) ainsi que les rayons issus des objets. Les points blancs correspondent aux pixels du plan de rendu commun, alors que les points rouges représentent les intersections des rayons. Les points verts correspondent quant à eux aux intersections situées à l'intérieur de la sphère et donc aux positions potentielles de la caméra pour l'image suivante. Enfin, les deux objets d'intérêt sont représentés par des cubes bleus et verts.

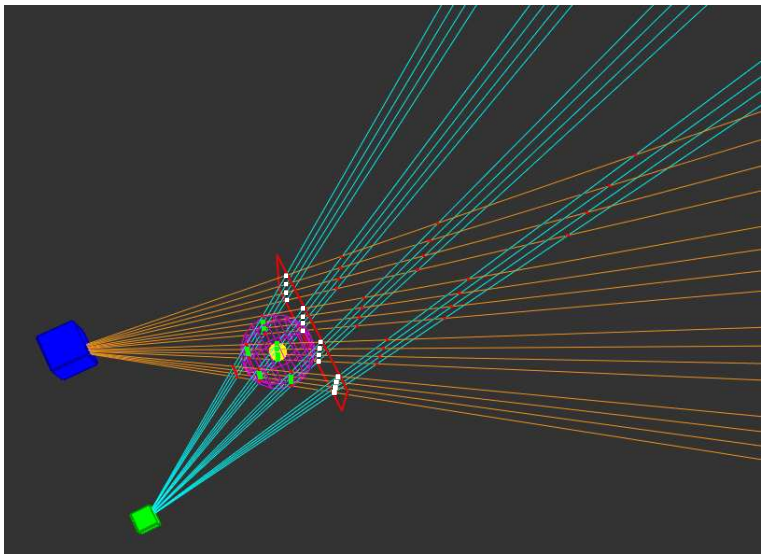


Figure 5.8 – Illustration du principe de l'OCCLUSION AVOIDER présentant les objets, le plan de rendu commun, les pixels de cet écran ainsi que les intersections calculées.

Une fois les rayons et les intersections calculés, nous effectuons le rendu depuis les objets vers le plan de rendu commun. Nous étudions ensuite les tampons de profondeur obtenus afin de caractériser les points verts (candidats à la prochaine position de caméra) en termes d'occlusion par rapport aux deux objets d'intérêt. À la suite de cette étape, la mise en relation d'une heuristique paramétrable et de l'accumulation des $i$ dernières images est prise en compte afin de déterminer l'intersection correspondante à la nouvelle position de caméra.

Dans la suite de ce chapitre, une section est consacrée à chacune des étapes de l'OCCLUSION AVOIDER, à savoir :

- la dynamique de la caméra,
- la construction des pyramides de vue, des plans de rendu et des intersections,
- les rendus depuis les objets,

- l'exploitation des rendus de profondeur en termes d'occlusion par rapport aux objets,
- l'accumulation des informations sur plusieurs *frames* consécutives et la mise en commun de ces informations,
- le choix du meilleur point du vue selon une heuristique.

### 5.2.1 Approximation des prochaines positions de caméra

La gestion de la dynamique de la caméra a pour but d'estimer les positions de caméra pour l'image suivante. Une fois ces positions approximées, nous pouvons déterminer les caractéristiques des plans et pyramides de vue, puis effectuer les rendus avant de déduire quelle position doit être choisie pour la caméra. L'approximation des futures positions de caméra est effectuée en deux étapes :

1. réduire l'espace de recherche des positions cohérentes de caméra à partir de la configuration actuelle (position, vitesse, accélération maximum autorisée) de manière à assurer une cohérence à l'écran, *i.e.* éviter des sauts de caméra, des changements trop brusques d'orientation, etc.
2. déterminer une distribution équitable des points de l'espace de recherche dans cet espace réduit.

La première étape consiste à déterminer l'ensemble des positions atteignables de caméra, en fonction de :

- sa position courante $p$,
- sa vitesse (un vecteur) $\vec{v}$,
- une valeur d'accélération maximale $a$, assurant une cohérence à l'image.

La solution choisie pour approximer ces positions est d'utiliser un schéma d'intégration d'Euler du premier ordre en fonction des paramètres de la caméra. L'approximation correspondante est peu précise, mais est obtenue de manière efficace. Nous utilisons les relations suivantes afin de caractériser la nouvelle position de caméra ($p(t+\delta_t)$) en fonction de la position courante ($p(t)$) et des paramètres décrits ci-avant :

$$v(t+\delta_t) = v(t) + \delta_t a(t) (\text{en effet, } a(t) = \dot{v}(t))$$

La position étant la dérivée de la vitesse, nous avons :

$$\begin{aligned} p(t+\delta_t) &= p(t) + \delta_t v(t+\delta_t) \\ &= p(t) + \delta_t v(t) + {\delta_t}^2 a(t) \end{aligned}$$

À chaque nouveau pas de temps de la scène, les positions des objets sont mises à jour en fonction de leurs trajectoires respectives et les nouvelles positions de caméra sont approximées par ce schéma d'intégration d'Euler. Afin de limiter les oscillations, nous avons également inclus un coefficient d'amortissement $c$ à l'équation. Le calcul de la prochaine position de caméra est donc réalisé comme suit :

$$p(t+\delta_t) = p(t) + (\delta_t v(t) + {\delta_t}^2 a(t))c \tag{5.2}$$

Afin de simplifier les approximations des positions futures de caméra, nous calculons la position à $t+\delta_t$ comme indiqué par l'équation 5.2 et approximons l'ensemble des positions atteignables par une sphère de centre $p(t+\delta_t)$ et de rayon correspondant à l'accélération maximale autorisée pour la caméra. Dans la suite de ce chapitre, nous faisons référence à cette sphère des futures positions de caméra comme étant la *sphère d'approximation* de la caméra.

La deuxième étape consiste en la génération d'une distribution équitable des positions à l'intérieur de la sphère d'approximation. Celle-ci est réalisée en fixant une densité de points correspondant à un

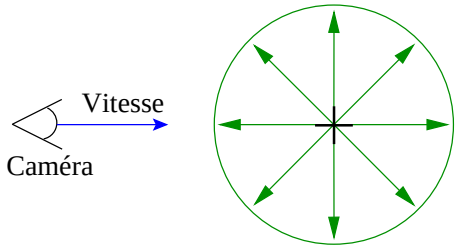



Figure 5.9 – Approximation des positions futures d'une caméra en mouvement par une sphère dont le rayon représente une accélération maximale uniforme : la sphère d'approximation.

nombre minimal d'intersections à calculer à l'intérieur de la sphère. La densité $d$ correspond au nombre de points par unité de volume de la sphère, le nombre d'intersections à générer à l'intérieur de la sphère est donc calculé comme suit :

$$nb_{points} = d * V$$

où $V$ représente le volume de la sphère de rayon $r$, c'est-à-dire :

$$V = \frac{4}{3}\pi r^3$$

L'utilisateur peut à tout moment modifier la valeur de la densité $d$ afin de générer plus ou moins de points dans la sphère et donc de modifier le nombre de candidats potentiels à la position suivante de caméra.

## 5.2.2 Calcul des pyramides, plans de vue, rayons et intersections

La sphère d'approximation permet de définir les pyramides de vue (également appelées *frustums*) utilisées lors du rendu de la scène depuis les objets. Ces pyramides sont calculées de façon à englober parfaitement la sphère d'approximation de la caméra, comme illustré en figure 5.10.

### *5.2.2.1 Calcul des* frustums *et des plans de vue*

Le calcul des pyramides et des plans de vue s'effectue en fonction des positions des objets et de la caméra, ainsi que du rayon de la sphère d'approximation. Les *frustums* sont construits de manière à être tangents à la sphère. La figure 5.11 illustre le schéma de construction des pyramides, dans lequel nous possédons les informations suivantes :

- la position $O$ d'un objet d'intérêt,
- la position du centre $C_S$ et la valeur $R_S$ du rayon de la sphère,
- la nature des triangles $OP_1C_S$ et $OP_2C_S$ : respectivement rectangles en $P_1$ et $P_2$,
- la position du centre $C_P$ du plan de rendu : le long de l'axe $OC_S$ et tangent à la sphère.

Nous pouvons dès lors calculer le centre $C_S$ de la sphère ainsi que les distances $OP_1$ et $OP_2$ :

$$C_P = C_S + (\overrightarrow{OC_S} \cdot R_S)$$

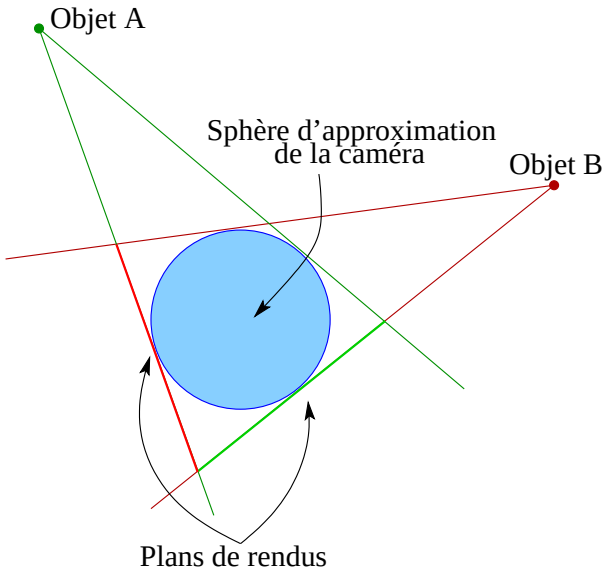



Figure 5.10 – Pyramides de vue tangentes à la sphère d'approximation de la caméra.

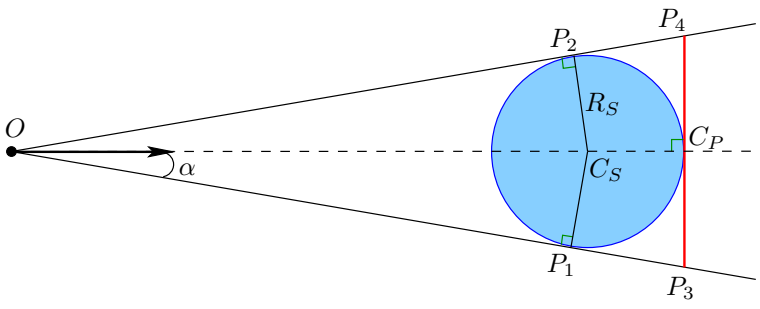



Figure 5.11 – Illustration de la construction d'une pyramide tangente à une sphère $S$.

et

$$\begin{aligned} OP_1 &= \sqrt{(OC_S)^2 - (C_SP_1)^2} \text{ avec } C_SP_1 = R_S(P_1 \in S) \\ OP_2 &= \sqrt{(OC_S)^2 - (C_SP_2)^2} \text{ avec } C_SP_2 = R_S(P_2 \in S) \end{aligned}$$

d'où

$$OP_1 = OP_2 = \sqrt{(OC_S)^2 - {R_S}^2} \text{ où } OC_S \text{ et } R_S \text{ sont connus.}$$

Sachant que $OP_1C_S$ (resp. $OP_2C_S$) est triangle en $P_1$ (resp. en $P_2$), nous pouvons déduire la valeur de l'angle $\alpha$ nous permettant de calculer la position de $P_1$ (resp. $P_2$), par la relation trigonométrique suivante :

$$\begin{aligned} \cos\alpha &= \frac{OP_1}{OC_S} = \frac{OP_2}{OC_S} \\ \cos\alpha &= \frac{\sqrt{(OC_S)^2 - {R_S}^2}}{OC_S} \end{aligned}$$

d'où

$$\alpha = \arccos\left(\frac{\sqrt{(OC_S)^2 - {R_S}^2}}{OC_S}\right)$$

Ces calculs nous permettent de définir les deux plans de rendu correspondants aux deux objets d'intérêt. Toutefois, nous souhaitons effectuer les rendus dans le même plan image, afin d'assurer que les rayons issus des objets d'intérêt et dirigés vers le plan de rendu s'intersectent. En effet, si les plans de rendus objets étaient différents, les rayons correspondants peuvent ne pas être coplanaires et ne pas présenter d'intersection. Ceci nous permet d'obtenir les informations concernant les occlusions le long des deux rayons à la fois.

Afin de fixer le même plan de rendu pour les deux objets, nous devons calculer les projections dans un même plan, des points tangents à la sphère à partir de chaque objet. Cette opération est illustrée en figure 5.12, nous remarquons ici que les *frustums* illustrés dans cette figure sont asymétriques, c'est-à-dire que le vecteur médian de la pyramide ne passe pas par le centre du rectangle de rendu commun.

Nous devons générer le plan de rendu qui englobe les projections obtenues depuis les objets $A$ et $B$. Cela consiste à calculer un rectangle qui en englobe deux autres, les trois rectangles appartenant au même plan $\mathcal{P}$. C'est ce plan qui nous sert de plan image pour les rendus issus des objets d'intérêt. La figure 5.13 illustre l'implémentation du calcul du plan de rendu commun dans l'application OCCLUSION AVOIDER.

Comme nous l'avons présenté plus haut, la sphère d'approximation est caractérisée par une densité représentant le nombre d'intersections qu'elle contient. Cette variable permet de contrôler la granularité de représentation des positions futures de la caméra et de caractériser la finesse de représentation. La densité nous permet de calculer les résolutions « horizontales » et « verticales » du plan de rendu. Pour ce faire, nous avons besoin des aires des plans images représentés dans la figure 5.13. Ce calcul s'effectue à partir du nombre d'intersections souhaité dans la sphère (cf. section 5.2.1), qui correspond exactement au cube ($x^3$) des résolutions verticales et horizontales de plans de rendu.

La figure 5.14 illustre le calcul des intersections de rayons issus de deux objets $A$ et $B$ et dirigés vers deux lignes de pixels, de résolution ($r = 5$) connue. Nous remarquons que le nombre d'intersections calculées correspond au carré de la résolution ($r^2$) de la ligne de rendu. Lorsque nous appliquons cette technique de calcul d'intersection non plus à une ligne mais à un plan de résolution carrée (*i.e.* dont

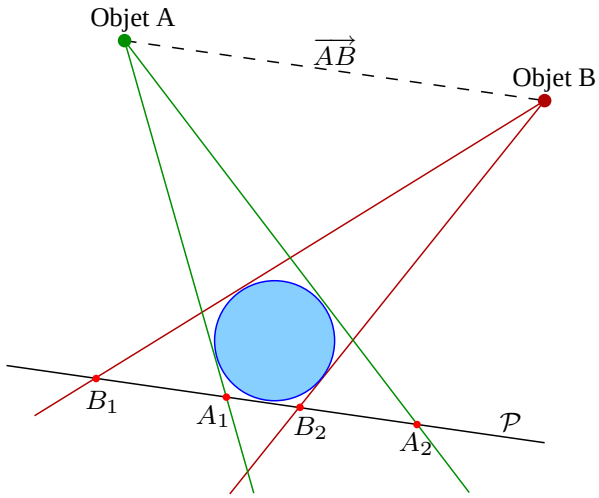



Figure 5.12 – Création de deux plans de rendu appartenant à un plan commun $\mathcal{P}$.

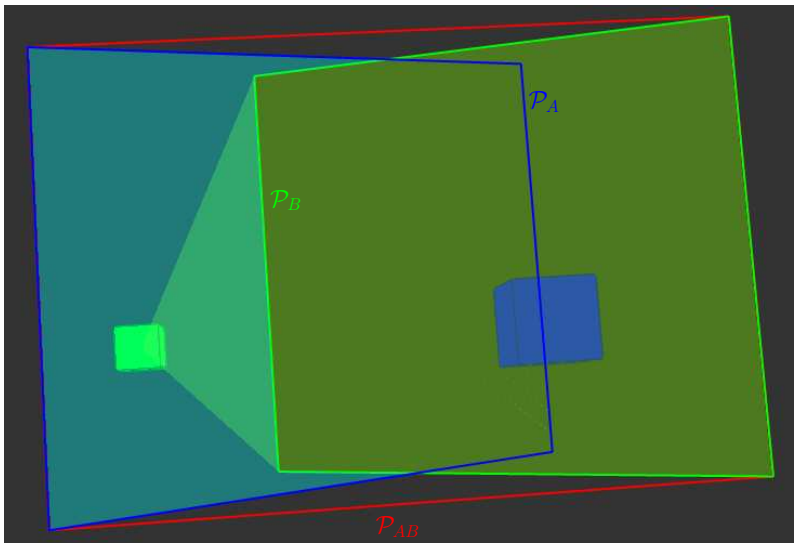



Figure 5.13 – Capture d'écran illustrant le calcul du plan de rendu commun dans l'OCCLUSION AVOIDER.

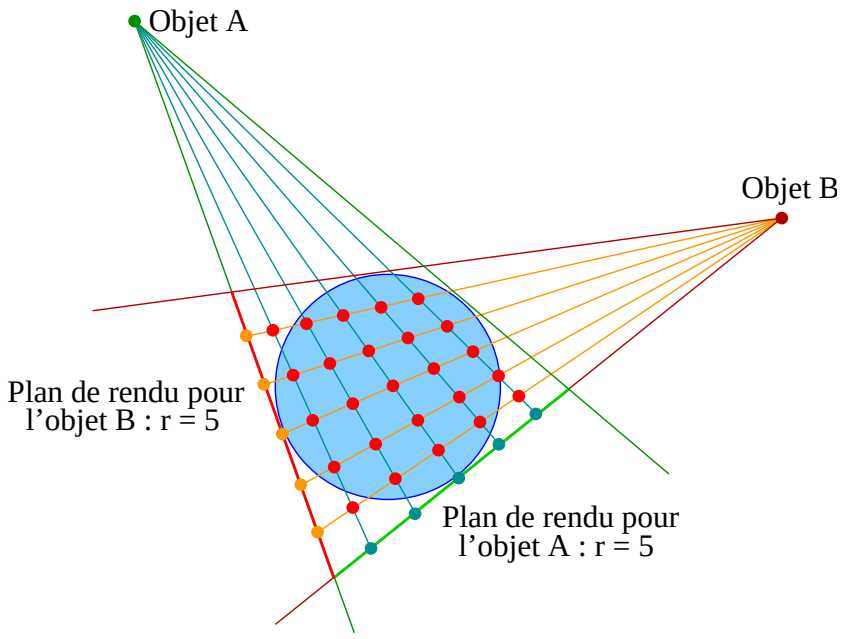



Figure 5.14 – Calcul du nombre d'intersections générées en fonction de la résolution (*i.e.* d'un nombre de rayons) du plan de rendu.

la résolution verticale est identique à la résolution horizontale), le nombre d'intersections calculées est $r^3$. Dans la suite de ce chapitre, pour des raisons de simplicité, nous considérons uniquement des plans de rendu à résolution carrée, c'est-à-dire ayant une résolution horizontale et verticale identique : $r_X = r_Y = r$.

En effet, la figure 5.14 correspond au calcul des intersections d'un plan à une hauteur $y$ fixe. Nous devons répéter ce calcul pour chacun des $y_i$ du plan de rendu (la composante verticale du plan), le nombre d'intersections générées est de $r^2$ pour chacune des hauteurs $y_i$. Puisque $r_X = r_Y = r$, nous sommes en présence de $r$ plans verticaux et avons ainsi généré $r^2 \times r = r^3$ intersections pour un plan 2D. La densité de la sphère d'approximation de la caméra nous permet de calculer la résolution $r$ des plans $\mathcal{P}_A$ et $\mathcal{P}_B$ de la manière suivante :

$$r = \sqrt[3]{d * V} \tag{5.3}$$

avec $d$ la densité de la sphère et $V$ son volume, cf. section 5.2.1.

Toutefois, à partir de l'information obtenue par l'équation 5.3, nous connaissons uniquement les résolutions des plans $\mathcal{P}_A$ et $\mathcal{P}_B$, il nous reste encore à calculer la résolution du plan $\mathcal{P}_{AB}$. Pour ce faire, nous utilisons les aires des plans $\mathcal{P}_A, \mathcal{P}_B$ et $\mathcal{P}_{AB}$. Les deux premiers plans contiennent chacun $r^2$ points (les pixels) pour une aire notée respectivement $\mathcal{A}_A$ et $\mathcal{A}_B$. La densité de pixels dans $\mathcal{A}_{AB}$ doit donc être au minimum celle du plus petit (en termes d'aire) des deux plans de rendus, c'est-à-dire la plus grande densité de pixels par unité d'aire. Pour déterminer la résolution du plan de rendu commun, nous calculons

une densité de pixels dans chacun des deux plans rendus des objets et dans le plan commun :

$$\begin{aligned} d_A &= \frac{r^2}{\mathcal{A}_A} \\ d_B &= \frac{r^2}{\mathcal{A}_B} \\ d_{AB} &= \mathsf{max}(d_A, d_B) \end{aligned}$$

Ces résolutions nous permettent de calculer le nombre de points $n_{AB}$ appartenant au plan de rendu commun, puis d'en déduire la résolution $r_{AB}$ associée :

$$\begin{aligned} n_{AB} &= \mathcal{A}_{AB} * d_{AB} \\ r_{AB} &= \sqrt{n_{AB}} \end{aligned}$$

La construction des *frustums* nécessaires aux rendus objets, ainsi que le calcul de l'équation et de la résolution du plan commun $\mathcal{P}_{AB}$ nous permettent de construire la grille de pixels utilisée pour les rendus dans les tampons de profondeur. La connaissance des coordonnées de ces pixels et des objets d'intérêt donnent lieu à la définition des rayons reliant chacun des objets à chaque pixel de l'écran de rendu commun. L'utilisation d'un plan unique de rendu pour les deux objets assure que les intersections de ces rayons forment une grille de points 3D, dont ceux appartenant à la sphère d'approximation correspondent aux points candidats pour les prochaines positions de caméra.

#### *5.2.2.2 Représentation des rayons et calcul d'intersections*

Une fois définies les coordonnées et la résolution du plan de rendu commun, nous pouvons calculer la position 3D des pixels appartenant à $\mathcal{P}_{AB}$ et déterminer les équations des rayons reliant les objets d'intérêt aux pixels. Un rayon $r(t)$ est représenté comme un point d'origine $O$ et un vecteur direction unitaire $\vec{d}$, un point étant localisé sur le rayon par l'expression suivante :

$$r(t) = O + t\vec{d} \tag{5.4}$$

Par convention, le vecteur $\vec{d}$ de l'équation 5.4 est unitaire. Le scalaire $t$ est utilisé pour définir les points le long du rayon. Enfin, puisque le vecteur direction est unitaire, le calcul de $r(t)$ génère un point sur le rayon situé à une distance de $t$ unités de l'origine $O$ du rayon.

Les positions des objets et les coordonnées 3D des pixels du plan de rendu commun étant connues, nous pouvons définir l'ensemble des rayons et générer la grille de points d'intersection illustrée en figure 5.8. L'intersection 3D $X = (x, y, z)$ de deux rayons $R_1$ et $R_2$ est calculée[1] et décrit en figure 5.15.

Nous considérons deux lignes $L_1$ et $L_2$ contenant respectivement les points 3D $X_1 = (x_1, y_1, z_1)$, $X_2 = (x_2, y_2, z_2)$, $X_3 = (x_3, y_3, z_3)$ et $X_4 = (x_4, y_4, z_4)$. L'existence du point $X(x, y, z)$, intersection des rayons $R_1$ et $R_2$, dépend de la condition de coplanarité des quatre points $X_1, X_2, X_3$ et $X_4$. Le point $X$ doit satisfaire à la fois les équations de chacun des deux rayons, à savoir :

$$\begin{aligned} X &= X_1 + \overrightarrow{(X_2 - X_1)} \cdot s \\ X &= X_3 + \overrightarrow{(X_4 - X_3)} \cdot t \end{aligned}$$

[1] Le calcul d'intersection ligne-ligne est décrit sur MathWorld à l'adresse : `http://mathworld.wolfram.com/Line-LineIntersection.html`

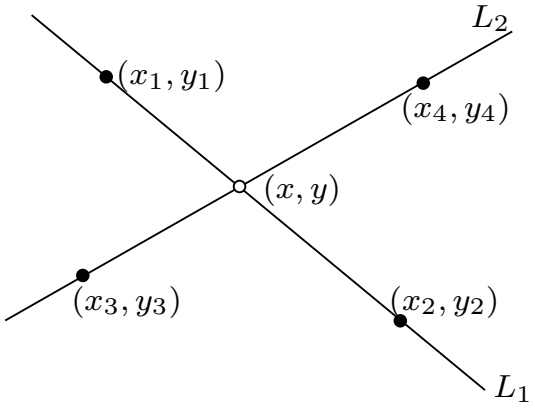



Figure 5.15 – Calcul de l'intersection entre deux droites 3D.

Sous réserve de coplanarité des quatre points appartenant aux rayons $R_1$ et $R_2$, l'intersection $X$ est calculée comme suit :

$$X = X_1 + \vec{a}\left(\frac{(\vec{c} \wedge \vec{b}) \cdot (\vec{a} \wedge \vec{b})}{\|\vec{a} \wedge \vec{b}\|^2}\right) \tag{5.5}$$

avec :

$$\begin{aligned} \vec{a} &= X_2 - X_1 \\ \vec{b} &= X_4 - X_3 \\ \vec{c} &= X_3 - X_1 \end{aligned}$$

L'ensemble des intersections est calculé en appliquant la formule 5.5. La grille de points ainsi formée est stockée ensuite à la fois en coordonnées Cartésiennes 3D et sous forme de système de coordonnées non affines, comme nous le présentons maintenant.

### 5.2.3 Représentation de la grille d'intersections par un système de coordonnées non affines

Une fois le plan de rendu commun $\mathcal{P}_{AB}$ déterminé pour les objets d'intérêt, nous pouvons calculer les rayons dont les intersections vont former une grille de points 3D (cf. figure 5.8). Nous pouvons déterminer le statut de l'occlusion concernant les deux objets pour chacun de ces points d'intersection, grâce à l'exploitation des rendus objets. En effet, les rayons allant des objets vers ces points d'intersection vont nous permettre de savoir si ceux-ci sont occultés ou non par des éléments de la scène. À la différence d'une méthode classique de *ray-casting*, qui calcule les intersections entre la géométrie de la scène et chacun des rayons, nous proposons d'utiliser des rendus matériels et de nous baser sur les informations de profondeur contenus dans ceux-ci pour déterminer le statut d'occlusion de chacun des points d'intersection.

Afin d'optimiser les calculs, la grille de points est stockée et calculée une seule fois par image. Nous pouvons déterminer la correspondance entre les points d'intersection, les rayons et les objets concernés en utilisant un système de coordonnées non affines. La caractérisation d'un point dans un repère non affine est différente de celle dans un repère Cartésien, comme illustré dans l'exemple suivant.

**Exemple 5.8 (Expression d'un point 2D en coordonnées Cartésiennes et en coordonnées non affines.).** *L'expression des coordonnées d'un point dans un système Cartésien 2D défini par la base orthonormée* $(\vec{i}, \vec{j})$ *est la suivante :*

$$O = x\vec{i} + y\vec{j} \tag{5.6}$$

*Exprimer un point* $P$ *dans un repère non affine de base* $(\overrightarrow{u_1}, \overrightarrow{u_2}, \overrightarrow{v_1}, \overrightarrow{v_2})$ *s'effectue de la manière suivante :*

$$P = \alpha\overrightarrow{u_1} + \beta\Big(\alpha\overrightarrow{v_1} + (1-\alpha)\overrightarrow{v_2}\Big) \tag{5.7}$$

Les rayons issus des objets d'intérêt et dirigés vers la grille de pixels du plan de rendu commun forment un repère non affine. La figure 5.16 illustre le processus d'intersection des rayons aboutissant à la définition de ce système non affine.

Chaque point d'intersection est défini en coordonnées globales 3D et l'est également dans l'un des repères non affines associé à un des objets d'intérêt, ce dernier est alors considéré comme objet de référence pour l'OCCLUSION AVOIDER. En effet, pour des raisons d'efficacité, nous stockons et caractérisons les intersections dans un seul des deux systèmes de coordonnées non affines. Étant donné un point $P$ en 3D et un ensemble de repères non affines, nous devons être capables de retrouver les coordonnées de $P$ dans chacun des repères. Le problème est illustré en figure 5.17. Le point d'intersection Inter de deux rayons est exprimé dans les repères locaux non affines définis pour les objets $A$ et $B$.

Nous illustrons dans la suite le calcul de $\alpha_2$ et $\beta_2$ permettant la localisation de Inter dans le repère non affine de l'objet $B$, les coordonnées $\alpha_1$ et $\beta_1$ sont obtenues de façon identiques pour l'objet $A$. Nous appelons dans la suite $P_{Inter}$ la projection du point Inter dans le plan de rendu associé au repère non affine de l'objet $B$. D'après l'équation 5.7, Inter est défini comme suit dans le repère attaché à l'objet $B$ :

$$P_{Inter} = \alpha_2\overrightarrow{u_{21}} + \beta_2((1-\alpha_2)\overrightarrow{v_{21}} + \alpha_2\overrightarrow{v_{22}} \tag{5.8}$$

Si nous développons l'équation 5.8, nous obtenons alors :

$$\begin{aligned} P_{Inter} &= \alpha_2\overrightarrow{u_{21}} + \beta_2(\overrightarrow{v_{21}} - \alpha_2\overrightarrow{v_{21}} + \alpha_2\overrightarrow{v_{22}}) \\ &= \alpha_2\overrightarrow{u_{21}} + \beta_2\overrightarrow{v_{21}} - \alpha_2\beta_2\overrightarrow{v_{21}} + \alpha_2\beta_2\overrightarrow{v_{22}} \\ &= \alpha_2(\overrightarrow{u_{21}} - \beta_2\overrightarrow{v_{21}} + \beta_2\overrightarrow{v_{22}}) + \beta_2\overrightarrow{v_{21}} \end{aligned}$$

soit :

$$P_{Inter} = \alpha_2(\overrightarrow{u_{21}} + \beta_2(\overrightarrow{v_{22}} - \overrightarrow{v_{21}})) + \beta_2\overrightarrow{v_{21}} \tag{5.9}$$

À partir de l'équation 5.9, nous pouvons exprimer la valeur de $\alpha_2$ en fonction des autres variables impliquées dans les relations, soit :

$$\alpha_2 = \frac{P_{Inter} - \beta_2\overrightarrow{v_{21}}}{\overrightarrow{u_{21}} + \beta_2(\overrightarrow{v_{22}} - \overrightarrow{v_{21}})} \tag{5.10}$$

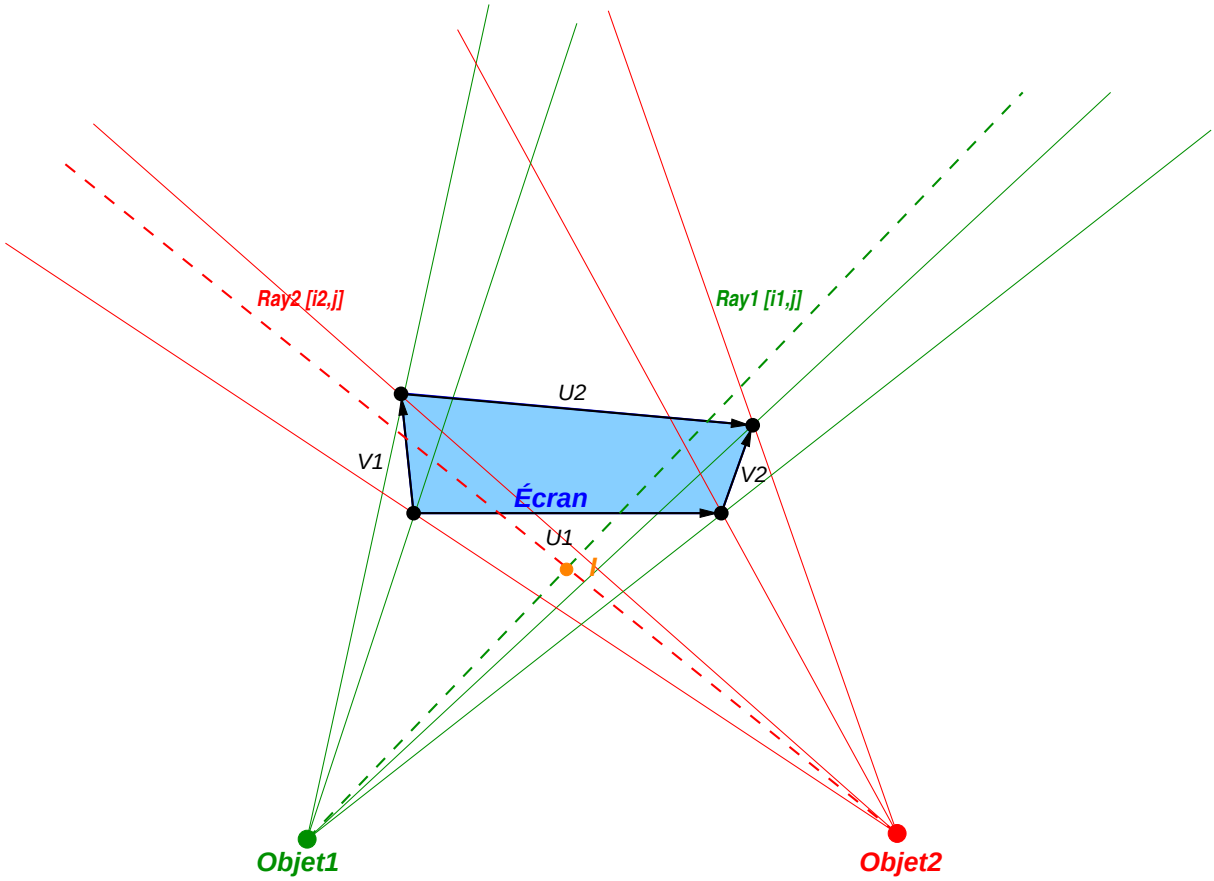



Figure 5.16 – Création de repères non affines à partir d'un ensemble de rayons issus des objets d'intérêt. Nous localisons les vecteurs $\vec{u_1}, \vec{u_2}, \vec{v_1}, \vec{v_2}$, ainsi que les quatre coins de l'écran. Un point d'intersection $I$ est également représenté en tant qu'intersection de deux rayons issus des objets d'intérêt $O_1$ et $O_2$.

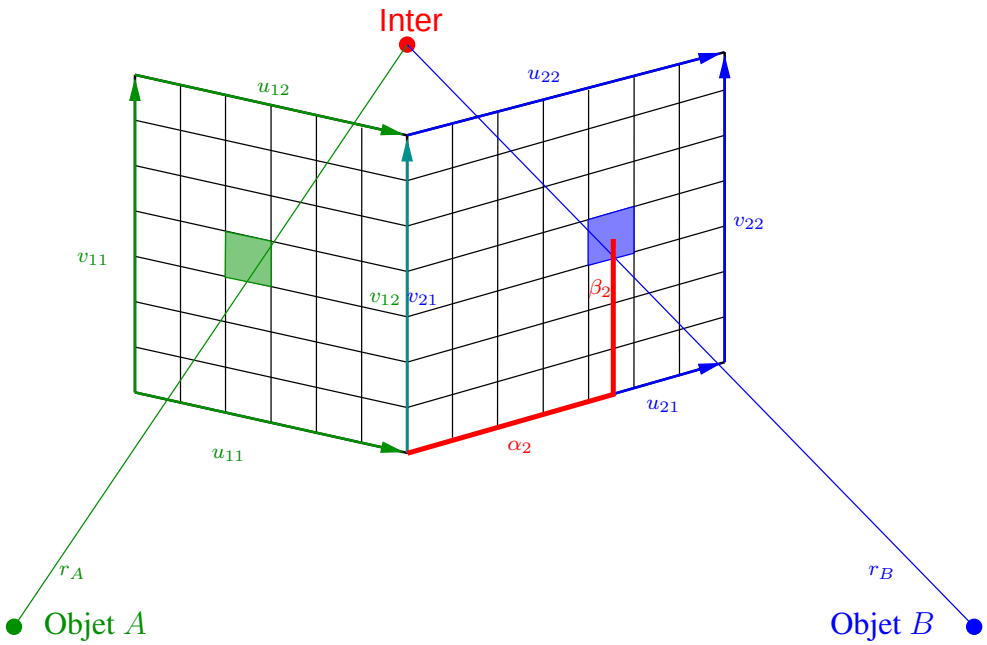



Figure 5.17 – Détermination des coordonnées des rayons $r_A$ et $r_B$ dans les systèmes non affines à partir des coordonnées globales du point d'intersection Inter de ces deux rayons.

Cette expression nous permet de remplacer $\alpha_2$ dans l'équation 5.9 et de calculer une valeur pour $\beta_2$. Pour ce faire, nous résolvons l'équation 5.9 en l'instanciant pour deux des trois coordonnées (*e.g.*, $X$ et $Y$) du point $P_{Inter}$, nous obtenons ainsi :

$$P_{InterY} = \frac{P_{InterX} - \beta_2 \overrightarrow{v_{21}}_X}{\overrightarrow{u_{21}}_X + \beta_2(\overrightarrow{v_{22}}_X - \overrightarrow{v_{21}}_X)} \Big( \overrightarrow{u_{21}}_Y + \beta_2(\overrightarrow{v_{22}}_Y - \overrightarrow{v_{21}}_Y) \Big) + \beta_2 \overrightarrow{v_{21}}_Y \tag{5.11}$$

En développant l'équation précédente, nous arrivons à la relation suivante :

$$\begin{aligned} P_{InterX}\overrightarrow{u_{21}}_Y - P_{InterY}\overrightarrow{u_{21}}_X = \beta_2 \Big( & P_{InterY}(\overrightarrow{v_{22}}_X - \overrightarrow{v_{21}}_X) - (\overrightarrow{u_{21}}_X \cdot \overrightarrow{v_{21}}_Y) - \\ & P_{InterX}(\overrightarrow{v_{22}}_Y - \overrightarrow{v_{21}}_Y) + (\overrightarrow{u_{21}}_Y \cdot \overrightarrow{v_{21}}_X) \Big) + \\ & {\beta_2}^2 \Big( \overrightarrow{v_{21}}_X(\overrightarrow{v_{22}}_Y - \overrightarrow{v_{21}}_Y) - \overrightarrow{v_{21}}_Y(\overrightarrow{v_{22}}_X - \overrightarrow{v_{21}}_X) \Big) \end{aligned} \tag{5.12}$$

L'équation 5.12 ainsi obtenue est une relation du second degré en $\beta_2$, de la forme :

$$a{\beta_2}^2 + b\beta_2 + c = 0 \tag{5.13}$$

avec :

$$\begin{aligned} a &= \overrightarrow{v_{21}}_X(\overrightarrow{v_{22}}_Y - \overrightarrow{v_{21}}_Y) - \overrightarrow{v_{21}}_Y(\overrightarrow{v_{22}}_X - \overrightarrow{v_{21}}_X), \\ b &= P_{InterY}(\overrightarrow{v_{22}}_X - \overrightarrow{v_{21}}_X) - \overrightarrow{u_{21}}_X\overrightarrow{v_{21}}_Y - P_{InterX}(\overrightarrow{v_{22}}_Y - \overrightarrow{v_{21}}_Y) + \overrightarrow{u_{21}}_Y\overrightarrow{v_{21}}_X, \\ c &= P_{InterY}\overrightarrow{u_{21}}_X - P_{InterX}\overrightarrow{u_{21}}_Y. \end{aligned}$$

Afin de résoudre ce type d'équation du second degré, il nous faut mener une étude de cas concernant les valeurs de $a, b$ et $c$.

**Si** $a = 0$ L'équation est alors de la forme $b\beta_2 + c = 0$ , ce qui correspond à une équation du premier degré, la solution de cette équation est donc déterminée de manière triviale comme étant : $\beta_2 = -c/b$.

**Si** $a \neq 0$ Alors nous devons résoudre l'équation 5.13 du second degré en calculant son discriminant :

$$\Delta \quad = \quad b^2 - 4ac$$

Là encore une étude de cas doit être menée pour calculer les solutions, en fonction du signe du discriminant $\Delta$. Trois cas de figure se présentent à nous :

$\Delta > 0$ l'équation possède alors 2 solutions :
- $\beta_{21} = \frac{-b-\sqrt{\Delta}}{2a}$
- $\beta_{22} = \frac{-b+\sqrt{\Delta}}{2a}$

Une seule des deux solutions $\beta_{21}$ ou $\beta_{22}$ est alors acceptable, celle ayant une valeur positive. Ces deux expressions ne pouvant être positives ou négatives en même temps, un simple test de signe nous fournit la valeur adéquate pour $\beta_2$.

$\Delta = 0$ l'équation ne possède alors qu'une seule solution : $\beta_2 = -b/(2a)$,

$\Delta < 0$ l'équation n'admet alors aucune solution dans l'espace des nombres réels. Cela signifie alors que le point $P_{Inter}$ n'est pas représentable dans le repère non affine étudié.

Une fois la valeur de $\beta_2$ déterminée, l'équation 5.10 permet de calculer une valeur pour $\alpha_2$, en instanciant arbitrairement une coordonnée de $P_{Inter}$ dans cette équation, *e.g.* la composante $X$ :

$$\alpha_2 = \frac{P_{Inter\,X} - \beta_2 \overrightarrow{v_{21}}_X}{\overrightarrow{u_{21}}_X + \beta_2(\overrightarrow{v_{22}}_X - \overrightarrow{v_{21}}_X)} \tag{5.14}$$

Les calculs précédents nous permettent d'exprimer les coordonnées de chacun des points d'intersection dans le repère non affine, tant en termes des coordonnées $\alpha$ et $\beta$, qu'en termes de coordonnées des rayons issus des objets. À l'inverse, nous sommes également capables, étant donné les coordonnées $[x_1, y_1]$ et $[x_2, y_2]$ de deux rayons d'inférer les coordonnées du point d'intersection correspondant (si celui-ci existe). Ainsi, en étudiant les valeurs des pixels des tampons de profondeur de chacun des objets, nous pouvons inférer le point d'intersection correspondant (cf. figure 5.16) et statuer sur son occlusion par rapport aux objets.

Avant d'étudier les rendus à partir des objets, nous donnons la représentation utilisée pour les systèmes de coordonnées non affines dans l'OCCLUSION AVOIDER. Pour chacun de ces systèmes, nous avons choisi de stocker les informations suivantes :
- le point 3D d'origine $O$ du système de coordonnées non affines situé à l'intersection des vecteurs $\vec{u_1}$ et $\vec{v_1}$,
- les quatre vecteurs $\vec{u_1}, \vec{u_2}, \vec{v_1}, \vec{v_2}$,
- les résolutions horizontales et verticales $r_H$ et $r_V$,
- l'équation du plan $\mathcal{P}$ contenant les pixels.

Ces informations sont illustrées dans la figure 5.17 et nous permettent d'effectuer toutes les opérations dont nous avons besoin sur ces systèmes de coordonnées non affines.

### 5.2.4 Effectuer les rendus objets

Afin de produire les tampons de profondeur permettant de déterminer le statut d'occlusion de chacun des points d'intersection, nous devons positionner et orienter une caméra sur chacun des objets d'intérêt. À l'instar de Halper et Olivier [HO00], les *depth buffers* que nous utilisons sont de résolution assez réduite, mais suffisante pour une détection correcte des occlusions. Toutefois, le nombre d'intersections calculées dans la sphère d'approximation est directement lié aux résolutions du plan de rendu commun. Nous devons par conséquent prendre en compte cette remarque lors de la définition des paramètres de construction de la sphère, afin de gérer le compromis entre efficacité (*i.e.* résolution réduite) et nombre d'intersections générées à l'intérieur de la sphère (*i.e.* augmenter le nombre de solutions potentielles du problème).

Nous devons positionner les *frustums* OPENGL afin de les centrer sur les objets d'intérêt et les orienter de manière à ce que le plan de rendu commun corresponde au plan éloigné (*far*) de la pyramide de vue. Les *frustums* OPENGL possèdent une orientation par défaut (l'axe $Z$ du repère monde) et sont définis par la fonction glFrustum qui possède six paramètres :

void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

Le volume contenu dans une pyramide de vue OPENGL est défini par les données des coordonnées 3D des coins bas gauche $(left, bottom, -near)$ et haut droit $(right, top, -near)$ du plan de coupe proche (*near clipping plane*), le plan éloigné étant parallèle au plan *near* et situé à une distance *far* de ce dernier, comme illustré en figure 5.18. Rappelons que les pyramides de vue utilisées sont asymétriques, c'est-à-dire que $left \neq -right$ et $bottom \neq -top$.

Une fois ces volumes construits, il faut les positionner sur les objets d'intérêt puis les orienter correctement. Ces opérations correspondent à quatre transformations OPENGL : une translation puis 3 rotations autour de chacun des axes du repère monde.

La translation est directe, il suffit de déplacer la pyramide selon les coordonnées de l'objet d'intérêt $O$, *i.e.* :

glTranslated(-$O_X$,-$O_Y$,-$O_Z$)

Pour chacune des trois rotations, il faut calculer un angle autour de chacun des axes $X, Y$ et $Z$. Pour ce faire, nous créons un repère $R(\overrightarrow{O_{V_x}}, \overrightarrow{O_{V_y}}, \overrightarrow{O_{V_z}})$ centré sur chacun des objets d'intérêt, tel que :

$$\begin{aligned} \overrightarrow{O_{V_x}} &= \overrightarrow{C_X} \\ \overrightarrow{O_{V_y}} &= C - O \\ \overrightarrow{O_{V_z}} &= \overrightarrow{O_{V_x}} \wedge (\overrightarrow{O_{V_y}} \wedge \overrightarrow{O_{V_x}}) \end{aligned}$$

Où $C$ représente la position de la caméra, $\overrightarrow{C_X}$ le vecteur $X$ du repère local du plan de rendu commun, c'est-à-dire le vecteur reliant les deux objets impliqués dans sa création, *i.e.* $\overrightarrow{C_X} = (O_2 - O_1)$. Notons $\overrightarrow{O_{V_y}}$ le vecteur reliant l'objet d'intérêt à la caméra. Il faut toutefois calculer le dernier vecteur $\overrightarrow{O_{V_z}}$ pour créer le repère. La figure 5.19 illustre le processus de calcul de ce vecteur. En effet, les vecteurs $\overrightarrow{O_{V_x}}$ et $\overrightarrow{O_{V_y}}$ ne sont pas nécessairement orthogonaux, il nous faut donc calculer deux produits vectoriels afin de déterminer la valeur de $\overrightarrow{O_{V_z}}$. Un premier produit vectoriel $\overrightarrow{O_{V_y}} \wedge \overrightarrow{O_{V_x}}$ nous fournit un vecteur temporaire $\overrightarrow{T}$ orthogonal à $\overrightarrow{O_{V_x}}$. Un deuxième produit vectoriel entre $\overrightarrow{O_{V_x}}$ et $\overrightarrow{T}$ détermine donc un troisième vecteur orthogonal à $\overrightarrow{O_{V_x}}$ qui crée un repère orthogonal : c'est le vecteur $\overrightarrow{O_{V_z}}$.

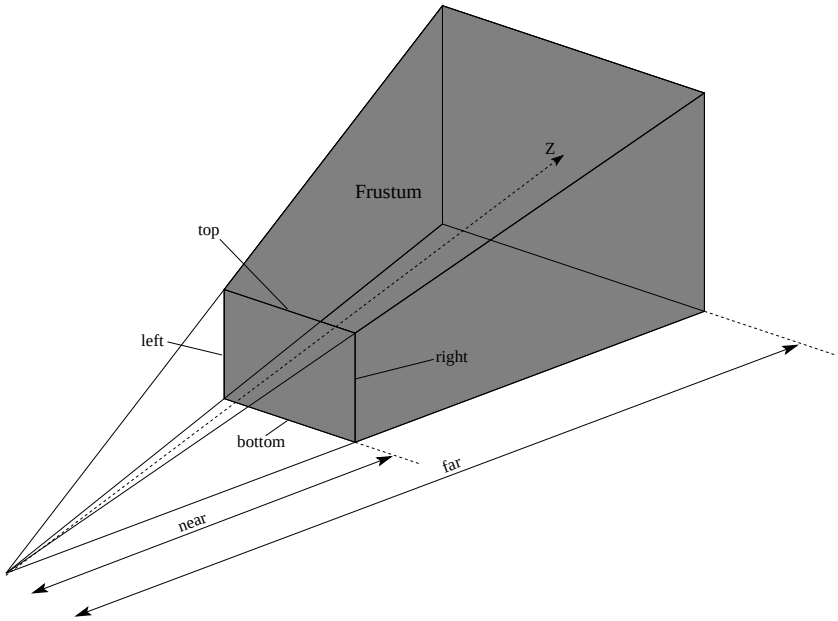



Figure 5.18 – Pyramide de vue OPENGL créée par un appel à la fonction glFrustum avec les paramètres $left, right, bottom, top, near$ et $far$, d'après Shreiner *et al.* [SWND05].

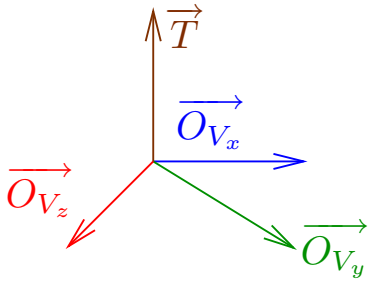



Figure 5.19 – Détermination de l'orientation d'un objet pour l'alignement de la pyramide de vue.

Afin d'orienter correctement le *frustum* OPENGL calculé pour chaque objet d'intérêt, il nous suffit d'aligner le vecteur $\overrightarrow{O_{V_z}}$ avec l'axe $Z$ du repère monde. Pour ce faire, nous devons calculer les angles $\theta, \phi$ et $\psi$ correspondants respectivement aux rotations autour des axes $X, Y$ et $Z$ du repère global, à partir des composantes du vecteur $\overrightarrow{O_{V_z}}$.

**Détermination de $\phi$, l'angle de rotation autour de l'axe $Y$** La valeur de $\phi$ est calculée grâce à la fonction atan2 appliquée aux composantes $X$ et $Z$ du vecteur orientation de l'objet $\overrightarrow{O_{V_z}}$ calculé précédemment :

$$\phi = \pi - \mathsf{atan2}(\overrightarrow{O_{V_z}x}, \overrightarrow{O_{V_z}z}) \tag{5.15}$$

**Détermination de $\theta$, l'angle de rotation autour de l'axe $X$** La valeur de la rotation autour de l'axe $X$ est déterminée par un calcul d'Arcsinus de la composante $Y$ du vecteur $\overrightarrow{O_{V_z}}$ :

$$\theta = -\arcsin(\overrightarrow{O_{V_z}y}) \tag{5.16}$$

**Détermination de $\psi$, l'angle de rotation autour de l'axe $Z$** Le calcul de la dernière rotation nécessite l'application des transformations concernant les angles $\phi$ et $\theta$ au vecteur $\overrightarrow{up}$ local du plan de rendu commun (*i.e.* $\overrightarrow{C_Y}$), ceci dans le but d'aligner le *frustum* OPENGL avec le plan de rendu commun. Nous supposons disposer ici des opérations rotationX$(\overrightarrow{V}, \alpha)$ et rotationY$(\overrightarrow{V}, \alpha)$ qui réalisent respectivement une rotation du vecteur $\overrightarrow{V}$ d'un angle $\alpha$ autour des axes $X$ et $Y$ du repère monde.

Le calcul de $\psi$ nécessite l'application de deux transformations au vecteur $\overrightarrow{up}$ du plan de rendu, puis par un calcul d'Arctangente :

$$\begin{aligned} \overrightarrow{T} &= \mathsf{rotationY}(\overrightarrow{C_Y}, \phi), \\ \overrightarrow{U} &= \mathsf{rotationX}(\overrightarrow{T}, \theta), \end{aligned}$$

nous avons alors :

$$\psi = \frac{\pi}{2} - \mathsf{atan2}(\overrightarrow{U}_y, \overrightarrow{U}_x). \tag{5.17}$$

**Rendu objet** La procédure nécessaire pour effectuer les rendus objets dans des tampons de profondeur OPENGL est la suivante, le code source correspondant est fournit dans la table 5.1 :

1. créer un cadre (*viewport*) OPENGL de dimension carrée identique à la résolution du plan de rendu (*i.e.* $r \times r$), par appel à la fonction glViewport$(0, 0, r, r)$,
2. définir la pyramide de vue asymétrique correspondant au plan de rendu commun grâce à la fonction glFrustum,
3. effectuer les transformations géométriques (translation et rotations) afin d'orienter et de positionner correctement le *frustum* sur l'objet d'intérêt,
4. effectuer le rendu dans un tampon de profondeur.

Les transformations OPENGL (rotations et translation) sont appliquées à l'envers de ce que nous aurions pu attendre, uniquement car OPENGL utilise une multiplication matricielle postfixée (les matrices

ALG. 5.1 – La méthode de rendu *offscreen* à partir d'un objet dans un tampon de profondeur OPENGL.

```
void offscreenRendering(Scene3D *s, Objet *o, DepthBuffer *dB)
{
// Paramétrage de la matrice de projection OpenGL
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

// définition d'une vue OpenGL
glViewport(0,0,m_resolH,m_resolV);

glFrustum(m_left,m_right,m_bottom,m_top,m_near,m_far);

// Paramétrage de la matrice de transformation de modèle OpenGL
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// Application des transformations OpenGL
// rotations autour de Z, X et Y puis la translation
glRotated(m_psi,0.,0.,1.);
glRotated(m_theta,1.,0.,0.);
glRotated(m_phi,0.,1.,0.);
glTranslated(-o->getX(),-o->getY(),-o->getZ());

// écriture dans le tampon arrière
glDrawBuffer(GL_BACK);

// élimination des faces avant
glCullFace(GL_FRONT);

// effacement des tampons de profondeut et de couleur
glClearColor(0.,0.,0.,0.);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// rendu de la scène spécifique pour les objets occultants
s->offscreenRendering();

// lecture dans le tampon arrière
glReadBuffer(GL_BACK);

// sauvegarde du rendu objet dans le tampon dB
glReadPixels(0,0,m_resolH,m_resolV,GL_DEPTH_COMPONENT,GL_FLOAT,dB);

// élimination des faces arrière
glCullFace(GL_BACK);
}
```

sont stockées en colonnes majeures). La méthode offscreenRendering d'un objet de type *Scene3D* permet de n'afficher que les objets nécessaires lors de la détermination de l'occlusion, c'est-à-dire les objets potentiellement occultants de la scène.

Enfin, la fonction OPENGL glReadPixels permet de copier un *buffer* de pixels de la mémoire graphique (*GPU*) vers la mémoire centrale (*CPU*). Ici, les paramètres spécifiés permettent de sauvegarder une image dont la taille correspond à celle du plan de rendu commun contenant les informations de profondeur (*GL_DEPTH_COMPONENT*) sous forme de nombres flottants (*GL_FLOAT*) et de l'écrire dans le *depth buffer* dB.

L'exécution de cette méthode pour chaque objet d'intérêt permet d'obtenir les informations de profondeur relatives aux occultants potentiels de la scène. L'analyse des *depth buffers* est présentée dans la section suivante.

### 5.2.5 Étude des tampons de profondeur

L'information de profondeur représente la complexité de la scène en fonction de la distance des objets par rapport à la caméra ayant effectuée le rendu. À chacun des pixels $p_i$ des tampons de profondeur est associée une information de profondeur $d_i$. OPENGL fournit une méthode de calcul de la valeur de profondeur dans la scène de l'objet projeté en $p_i$ en fonction de $d_i$. Cette relation s'exprime en fonction des distances des plans de *clipping* près ($z_{near}$) et éloigné ($z_{far}$) du *frustum* utilisé pour effectuer le rendu *offscreen*. La relation est la suivante :

$$\begin{aligned} b &= (z_{far} * z_{near})/(z_{near} - z_{far}) \\ a &= z_{far}/(z_{far} - z_{near}) \end{aligned}$$

La valeur de profondeur $z_i$ associée à $d_i$ est alors obtenue comme suit :

$$z_i = b/(d_i - a) \tag{5.18}$$

Cette valeur $z_i$, obtenue grâce à la valeur $d_i$ stockée dans le tampon de profondeur, nous permet d'étudier la profondeur de l'occultant le plus proche par rapport à l'objet d'intérêt. En effet, si la valeur $z_i$ de l'occultant est supérieure à la distance entre le point d'intersection et l'objet d'intérêt $O_i$, alors l'occultant se trouve derrière l'objet et le point d'intersection doit être considéré comme solution potentielle au problème d'occlusion pour $O_i$. Sinon, l'occultant est situé entre l'objet d'intérêt et le point d'intersection, ce dernier est alors occulté par rapport à $O_i$.

Un raisonnement similaire est appliqué pour les tampons de profondeur de chacun des objets d'intérêt $O_i$ et le point d'intersection est alors considéré comme solution au problème global si il est solution pour chacun des objets d'intérêt $O_i$. L'étude de chaque point d'intersection nécessite le maintien à jour d'une structure de données au sein de l'OCCLUSION AVOIDER.

### 5.2.6 Stockage des informations de profondeur

Afin de pouvoir déterminer les statuts d'occlusion de chaque point d'intersection des rayons, nous avons mis en place une structure de données permettant de stocker pour chacun d'eux un ensemble d'informations relatives aux deux objets d'intérêt et à la caméra, à savoir :

- une position 3D $P$,
- la distance à la position courante de caméra $d_C$,
- la distance $d_A$ par rapport à l'objet $A$,

- la distance $d_B$ par rapport à l'objet $B$,
- un booléen $O_A$ caractérisant l'occlusion par rapport à l'objet $A$,
- un booléen $O_B$ caractérisant l'occlusion par rapport à l'objet $B$,
- un booléen $O_{AB}$ caractérisant l'occlusion par rapport aux deux objets $A$ et $B$ (*i.e.* le ET booléen des deux valeurs précédentes et stocké pour des raisons d'efficacité),
- les coordonnées $(x_A, y_A)$ du rayon issu de l'objet $A$ correspondant à ce point d'intersection,
- les coordonnées $(x_B, y_B)$ du rayon issu de l'objet $B$ correspondant à cette intersection.

La position 3D $P$ représente les coordonnées de l'intersection en coordonnées globales, les distances à la caméra et aux objets d'intérêt permettent de discriminer les intersections équivalentes en termes d'occlusion par rapport à un ensemble d'heuristiques proposées. Nous pouvons en effet choisir par exemple de privilégier les positions proches de la position courante de la caméra afin de maintenir une cohérence à l'image, c'est-à-dire de limiter les différences entre deux *frames* successives en limitant les déplacements de caméra.

Les booléens représentent le statut de l'intersection courante en termes d'occlusion par rapport aux objets d'intérêt. Par exemple, si le booléen $O_A$ vaut la valeur *vrai*, cela signifie que cette intersection n'est pas occultée par rapport à l'objet $A$. Ainsi, ce point est candidat à la prochaine position de caméra si l'on souhaite assurer la visibilité de $A$ à l'écran.

Enfin, les distances $d_A$ et $d_B$ par rapport aux objets permettent de déterminer si les points d'intersection sont situés devant ou derrière les occultants potentiels de la scène. La figure 5.20 illustre la nécessité de stocker ces distances par rapport aux objets d'intérêt. En effet, nous effectuons les rendus de la scène depuis les objets dans des tampons de profondeur (*depth buffers*), c'est-à-dire que les valeurs stockées dans les pixels ne sont pas des couleurs mais les distances entre un occultant potentiel et la caméra effectuant le rendu (positionnée sur l'objet d'intérêt). Nous devons donc déterminer si chaque point d'intersection est situé derrière un des occultants de la scène : dans ce cas, ce point doit être considéré comme une solution potentielle. En effet, si la caméra est déplacée à cette position, les deux objets apparaissent à l'écran sans occlusion.

Sur la figure 5.20, un des points solutions illustre cette notion d'information de profondeur : l'intersection des deux rayons issus des objets $A$ et $B$ mis en évidence (plus épais) sur la figure. En effet, le rayon issu de l'objet A ne rencontre aucun occultant dans la scène, ce point est non occulté pour cet objet. Au contraire, le rayon issu de l'objet B rencontre un objet dans la scène, l'intersection doit donc être considérée comme occultée pour B. Cependant, nous voyons que l'occultant est derrière ce point, à l'intérieur de la sphère d'approximation. Ainsi, en utilisant l'équation 5.18, nous déterminons que ce point d'intersection est également non occulté pour l'objet $B$, il doit donc être considéré comme solution potentielle au problème et éligible à la prochaine position de caméra.

### 5.2.7 Accumulation des informations de profondeur au cours du temps

Afin d'assurer une cohérence à l'image (éviter les sauts de caméra, etc.) et de tirer profit des déplacements précédemment effectués dans la scène, nous stockons les informations de profondeur pour chacun des points d'intersection pour un certain nombre d'images consécutives. Cette agrégation d'informations nous permet de prendre en compte une continuité temporelle dans la scène. Ainsi, lors du choix du meilleur point de vue pour l'image suivante, nous consultons l'historique des occlusions précédentes, en interpolant les valeurs obtenues pour les intersections proches (rappelons que la position $P$ d'une intersection est stockée en coordonnées globales). Cette agrégation nous permet de ne pas réagir à des événements très succincts dans le temps.

Prenons l'exemple d'un objet se déplaçant dans la scène 3D et occultant un des objets d'intérêt pour

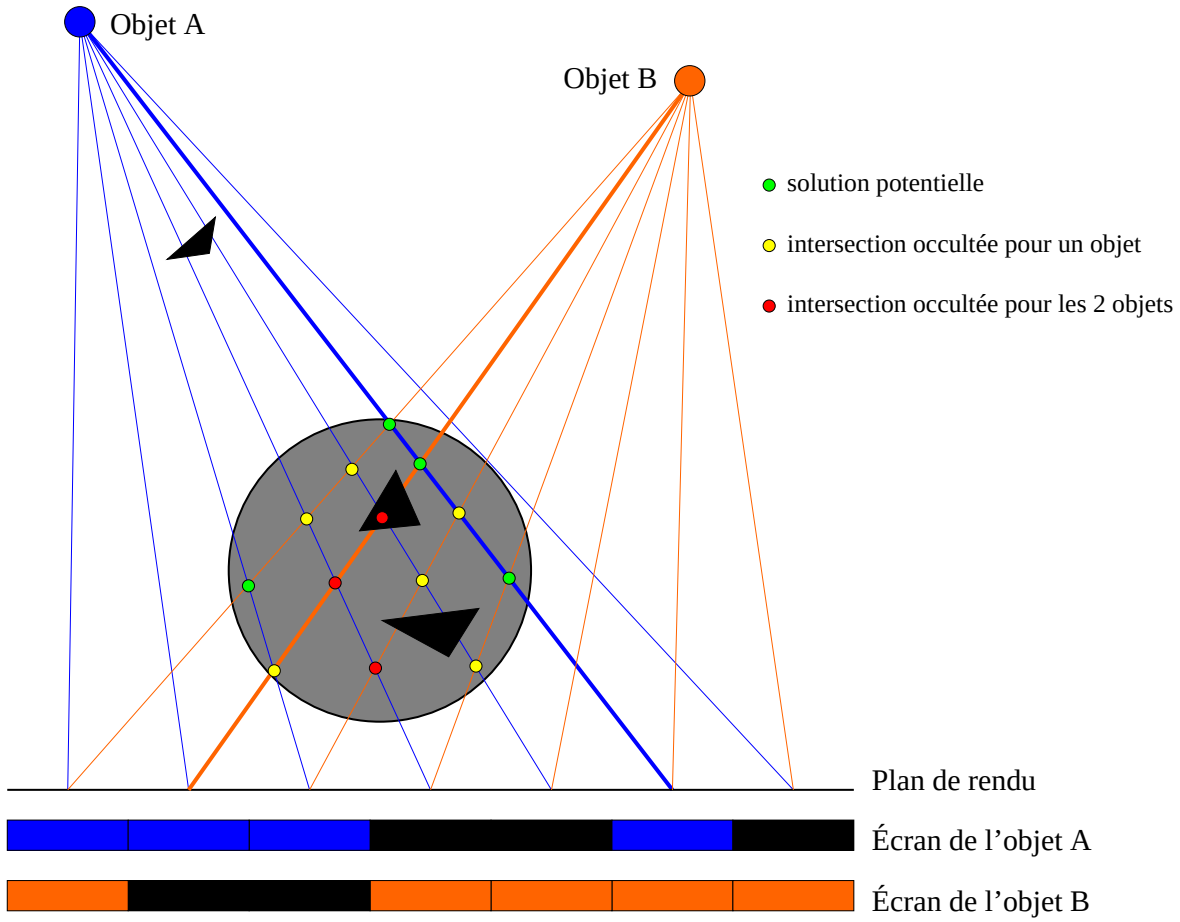



Figure 5.20 – Création des « rayons » issus des objets pour le rendu de la scène vers la sphère des positions futures de caméra. Les statuts d'occlusion des intersections sont gérés en prenant en compte les informations de profondeur des *depth buffers* afin de déterminer s'ils sont ou non situés devant les occultants et correspondent à une solution potentielle.

une seule image. Si ne nous tenons pas compte de l'historique des occlusions en plus d'une heuristique de choix pour la prochaine position de caméra, la position courante peut être considérée comme inadéquate puisqu'au moins un des deux objets d'intérêt est occulté pour la *frame* courante. Toutefois, l'occlusion étant très succincte (une seule image), modifier la position de la caméra provoque alors une incohérence à l'écran. En effet, un déplacement de caméra modifie la représentation de la scène à l'image, plus préjudiciable à la perception de l'environnement par l'utilisateur qu'une occlusion très temporaire d'un des objets d'intérêt.

Ainsi, nous proposons d'intégrer au processus de choix de la nouvelle position de caméra, basé sur une heuristique, l'historique des occlusions des points candidats lors des $i$ dernières *frames*. Ceci nous permet de ne pas éliminer des points occultés pour l'image courante si ceux-ci satisfont le critère de l'heuristique choisie (par exemple minimiser le déplacement de caméra afin d'assurer une cohérence à l'écran).L'implémentation actuelle de l'OCCLUSION AVOIDER considère une valeur $i$ égale à $5$ images. Afin de décider si une position est éligible ou non, nous récupérons son statut d'occlusion pour les $5$ dernières *frames*.

Étant donné la forte dynamique de la scène, les points candidats correspondants aux intersections des rayons n'ont que de très faibles chances de se trouver consécutivement exactement aux mêmes coordonnées. Afin de résoudre ce problème, nous stockons les points candidats en coordonnées globales et mettons ces valeurs à jour à chaque pas de rafraîchissement de l'application pour limiter des écarts trop importants. Ensuite, lors de l'analyse d'un point candidat, nous parcourons les grilles des images précédentes pour trouver les informations correspondantes aux points les plus proches (en coordonnées globales) étudiés précédemment. Nous agrégeons ainsi de l'information passée lors de l'analyse du point courant.

**Exemple 5.9 (Sélection d'un point candidat).** *Soit $P$ la position étudiée couramment, correspondant au point d'intersection des rayons $[1, 2]$ pour l'objet $A$ et $[2, 3]$ pour l'objet $B$.*
*Afin de décider si $P$ est candidat à la prochaine position de caméra, nous consultons l'historique des points d'intersection de coordonnées $[1, 2]$ et $[2, 3]$ pour les $5$ dernières images. Nous consultons ensuite les booléens associés à ces points afin de vérifier le statut d'occlusion de chacun d'eux par rapport aux objets d'intérêt. Si le nombre d'images pour lesquelles ces points aboutissent à une vue non occultée des objets d'intérêt est supérieur à un seuil (modifiable dans* OCCLUSION AVOIDER*), alors $P$ est ajouté à la liste des candidats potentiels, sinon $P$ est écarté.*

Afin de tirer profit de cette accumulation des informations de profondeur au cours du temps nous devons être capables, étant donné les coordonnées 3D globales d'un point d'intersection $P$, de trouver les points correspondants dans les *frames* précédentes.

#### *5.2.7.1 Mise en correspondance de deux systèmes de coordonnées non affines*

L'accumulation des informations d'occlusion pour les deux objets d'intérêt nous permet d'intégrer une dimension temporelle lors de l'évaluation des intersections courantes. L'idée consiste, étant donné une intersection en 3D représentée dans un système de coordonnées non affines, à trouver dans les *frames* précédentes les points les plus proches et à prendre en compte les occlusions correspondantes. Ainsi, une intersection non occultée dans quatre images consécutives est considérée plus intéressante qu'une intersection qui l'a été seulement deux fois. En fixant une limite sur le nombre de frames consécutives lors du choix des points candidats, nous pouvons prendre en compte les cas pathologiques, comme l'exemple d'un objet passant très succinctement devant la caméra.

Afin de prendre en compte les informations accumulées lors des images précédentes, un protocole

est mis en place lors de l'évaluation de chaque intersection courante. En fonction de la position dans le repère 3D global de l'intersection $I$, des positions des deux objets d'intérêt $O_1$ et $O_2$, ainsi que des deux systèmes de coordonnées non affines associés $\mathcal{S}_1$ et $\mathcal{S}_2$, nous devons suivre la méthodologie présentée en sous-section 5.2.3 et illustrée en figure 5.17, *i.e.* :

- calculer les coordonnées 3D de la projection $P_{O_1}$ du rayon $(O_1I)$ sur le plan $\mathcal{P}_1$ du système de coordonnées non affines $\mathcal{S}_1$ de l'objet $O_1$,
- calculer les coordonnées 3D de la projection $P_{O_2}$ du rayon $(O_2I)$ sur le plan $\mathcal{P}_2$ du système de coordonnées non affines $\mathcal{S}_2$ de l'objet $O_2$,
- déterminer les coefficients $\alpha_1$ et $\beta_1$ permettant de représenter $P_{O_1}$ dans le système $\mathcal{S}_1$,
- déterminer les coefficients $\alpha_2$ et $\beta_2$ permettant de représenter $P_{O_2}$ dans le système $\mathcal{S}_2$,
- convertir les composantes $(\alpha_1, \beta_1)$ en résolutions $(x_1, y_1)$ du point $P_{O_1}$ dans le système $\mathcal{S}_1$,
- convertir les composantes $(\alpha_2, \beta_2)$ en résolutions $(x_2, y_2)$ du point $P_{O_2}$ dans le système $\mathcal{S}_2$,
- déterminer si $P_{O_1}$ est effectivement représentable dans le système $\mathcal{S}_1$,
- déterminer si $P_{O_2}$ est effectivement représentable dans le système $\mathcal{S}_2$,
- récupérer les statuts d'occlusion $o_i$ des points précédents correspondants à $I$ dans la structure de données,
- modifier l'évaluation du point courant $I$ en fonction du statut $o$ dans les images précédentes.

Nous détaillons brièvement chacune de ces étapes.

**Calcul de la projection de $I$ sur le plan du système $\mathcal{S}_1$** Les coordonnées $(\alpha_i, \beta_i)$ de la projection d'un point d'intersection $I$ sur le plan $\mathcal{P}_i$ associé au repère $\mathcal{S}_i$ du point d'intérêt $O_i$ sont calculées en construisant un rayon $R$ reliant l'objet $O_i$ à $I$. Grâce aux équations du rayon $R$, du plan $\mathcal{P}_i$ et aux calculs décrits en section 5.2.3 dans la figure 5.17, nous calculons les coordonnées $(\alpha_i, \beta_i)$ correspondant à la projection $P_{O_i}$ de $I$ sur $\mathcal{P}_i$.

**Calcul des coordonnées des projections de $I$ dans les systèmes $\mathcal{S}_1$, $\mathcal{S}_2$ et en termes de pixels des *depth buffers*** Une fois les coordonnées globales $(\alpha_1, \beta_1)$ et $(\alpha_2, \beta_2)$ des points $P_{O_1}$ et $P_{O_2}$ déterminées, nous devons calculer les indices des rayons correspondants. Les deux valeurs $(\alpha_i, \beta_i)$ correspondant aux projections $P_{O_i}$ et calculées à l'étape précédente, appartiennent à l'intervalle de réels $[0, 1]$. Nous devons les convertir en $(x_i, y_i)$ indices des rayons correspondants et appartenant à l'intervalle $[0, r]$, où $r$ représente la résolution des plans de rendu. Cette conversion est illustrée en figure 5.17.

**Représentation des points dans un système de coordonnées non affines** La section 5.2.3 décrit comment déterminer si un point est représentable ou non à l'intérieur d'un système de coordonnées non affines. Nous appliquons le même processus pour déterminer si les points $P_{O_1}$ et $P_{O_2}$ sont représentables dans les systèmes $\mathcal{S}_1$ et $\mathcal{S}_2$.

**Agrégation des informations précédentes** Le statut d'occlusion du point $I$ est mis à jour après consultations de l'historique des occlusions aux images précédentes. Pour ce faire, nous consultons notre matrice d'accumulation pour chacune des images précédentes au point d'intersection des rayons $R_1[x_1, y_1]$ et $R_2[x_2, y_2]$ dont les coordonnées ont été calculées dans les étapes précédentes.

### 5.2.8 Choix du meilleur point de vue

À ce stade de l'application, nous avons déterminé le statut d'occlusion par rapport à chacun des objets d'intérêt pour chacune des intersections candidates (c'est-à-dire situées à l'intérieur de la sphère d'approximation de la caméra). Nous devons dès lors départager les candidats *ex aequo* afin de choisir la prochaine position de caméra. Pour ce faire, nous avons recours à une heuristique permettant de déterminer quelle intersection est la plus avantageuse lorsque plusieurs d'entre elles conduisent à une vue non occultée de tous les objets d'intérêt, c'est-à-dire lorsque les configurations correspondent à des solutions équivalentes en termes d'évitement d'occlusions.

Nous avons implémenté plusieurs heuristiques afin d'étudier leur impact sur les solutions obtenues :

- minimiser la distance entre le candidat et la position actuelle de caméra,
- minimiser le déplacement de la caméra par rapport à l'objet $O_1$,
- minimiser le déplacement de la caméra par rapport à l'objet $O_2$,
- minimiser le déplacement de la caméra par rapport au point médian des deux objets d'intérêt,
- conserver la distance par rapport à l'objet $O_1$ dans un intervalle de valeurs $[d_1, d_2]$,
- conserver la distance par rapport à l'objet $O_2$ dans un intervalle de valeurs $[d_1, d_2]$,
- conserver la distance par rapport au point médian des deux objets d'intérêt dans un intervalle de valeurs $[d_1, d_2]$.

Afin de limiter les mouvements de caméra entre deux *frames* successives et de maintenir une cohérence à l'image, nous avons défini les heuristiques contraignant les positions de caméra à l'intérieur d'un intervalle de valeurs limitant les déplacements de caméra. Enfin, si aucune heuristique n'aboutit à une solution satisfaisante ou si aucun candidat ne peut conduire à une vue non occultée des objets d'intérêt, nous décidons de ne pas déplacer la caméra.

## 5.3 L'outil OCCLUSION AVOIDER

La méthode présentée dans ce chapitre est implémentée au sein du prototype OCCLUSION AVOIDER, développé en C++ et OPENGL. La figure 5.21 illustre une capture d'écran de cette application, dans laquelle sont affichés le plan de rendu commun, les pixels de l'écran associé ainsi que les rayons issus des deux objets d'intérêt $O_1$ et $O_2$. De plus, le prototype propose différents modes de vue, nous avons choisi de diviser l'écran en quatre *viewports* :

- le premier présente une vue 3D de la scène et permet à l'utilisateur de naviguer au sein de l'environnement,
- le cadre situé en bas à gauche met en évidence la vue obtenue depuis la caméra, nous voyons donc ici les rayons issus des objets d'intérêt,
- le *viewport* situé au milieu et en bas de l'application propose une vue depuis le premier objet d'intérêt $O_1$ (le cube vert de la scène 3D),
- la dernière vue affiche quant à elle la scène depuis le deuxième objet d'intérêt $O_2$ (le cube bleu).

Les deux dernières vues affichent également les pixels correspondant aux rendus objets, ces derniers permettent de visualiser quelle partie des tampons de profondeur contiennent des occultants potentiels. Les pixels qui contiennent des points d'intersection appartenant à la sphère d'approximation permettent de visualiser si les pixels des depth buffers contiennent une valeur ou sont vides. Les trois figures 5.22, 5.23 et 5.24 illustrent notre approche. Les intersections correspondantes aux solutions potentielles pour les prochaines positions de caméra sont affichées selon un code couleur :

- vert : si l'intersection aboutit à une vue non occultée des deux objets d'intérêt,
- turquoise : si l'intersection est occultée vis-à-vis de l'objet $O_1$,

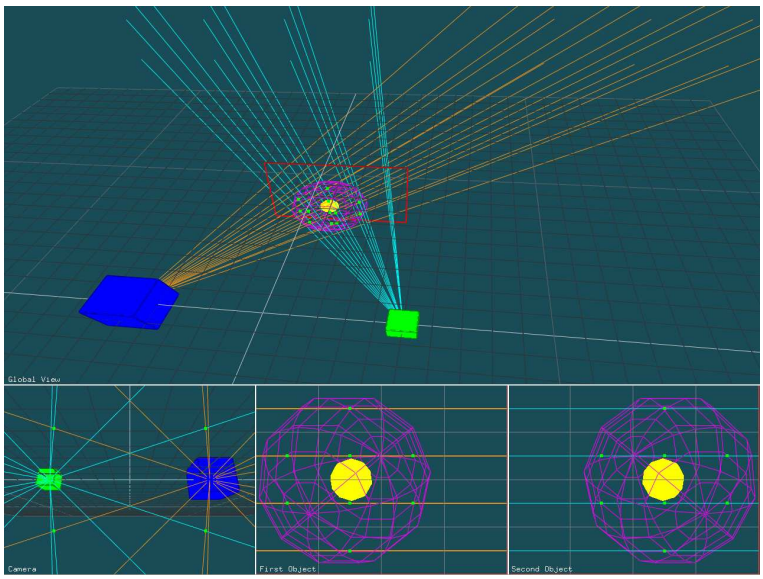



Figure 5.21 – Capture d'écran de l'OCCLUSION AVOIDER.

- jaune : si l'intersection est occultée pour l'objet $O_2$,
- turquoise : si l'intersection est occultée pour les deux objets d'intérêt $O_1$ et $O_2$.

## 5.4 Résultats

## 5.5 Choix du meilleur point de vue pour un modèle 3D

Dans l'implémentation actuelle de l'OCCLUSION AVOIDER, les rendus objets sont effectués depuis les barycentres des objets d'intérêt. Toutefois, le choix du centre de l'objet est critiquable et de meilleures alternatives peuvent être mises en œuvre. Dans ce but, nous avons développé une application indépendante (MODELVIEWER) qui permet de calculer un ensemble de points de projection représentatifs d'un modèle 3D en fonction d'une position de caméra. Cette application a pour but de générer, lors d'un prétraitement, une structure de données stockant l'ensemble des meilleurs points de projection en fonction de la position courante de caméra. Le principe du MODELVIEWER consiste, après avoir chargé un modèle 3D, à définir un vecteur représentant l'orientation intrinsèque de l'objet. Nous générons ensuite un ensemble de positions de caméra autour de cet objet dans l'espace 3D. Depuis chacune de ces positions nous effectuons un rendu colorimétrique de l'objet dans un tampon image (*frame buffer*) pour lequel chacune des facettes du modèle étudié est affichée avec une couleur unique et en l'absence de source lumineuse. Une fois ce rendu effectué, nous parcourons le *frame buffer* et déterminons la facette la plus représentée à l'image : celle qui correspond à la couleur apparaissant le plus souvent dans le tampon image. Le polygone correspondant est donc le plus représentatif de l'objet depuis le point de vue courant, son barycentre correspond ainsi à un point de projection caractéristique pour la position courante de caméra.

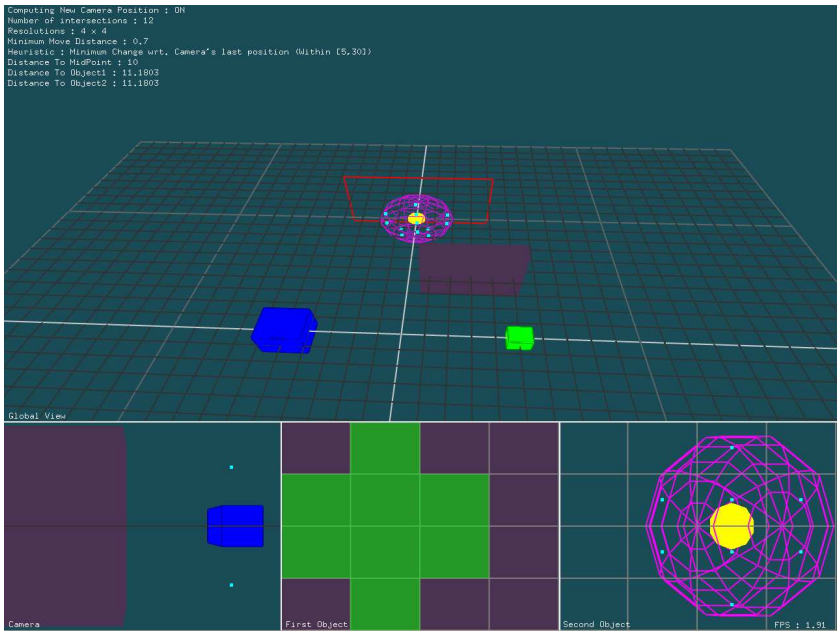



Figure 5.22 – Illustration d'une occultation de l'objet $O_1$, représenté par un cube vert, dans le prototype OCCLUSION AVOIDER. Les points d'intersection appartenant à la sphère d'approximation sont affichés en bleu turquoise pour signaler que l'objet $O_1$ est occulté.

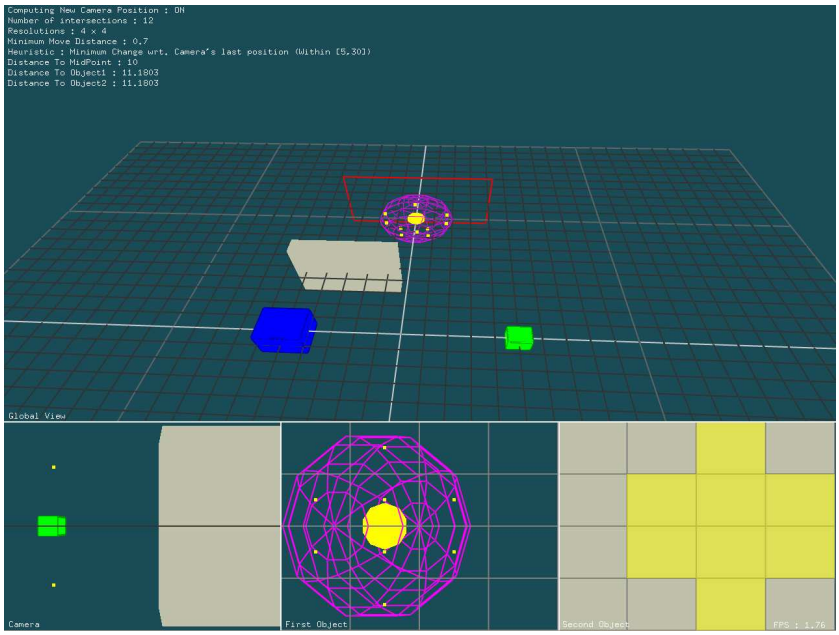



Figure 5.23 – Illustration d'une occultation de l'objet $O_2$, représenté par un cube bleu, dans le prototype OCCLUSION AVOIDER. Les points d'intersection appartenant à la sphère d'approximation sont affichés en jaune pour signaler que l'objet $O_2$ est occulté.

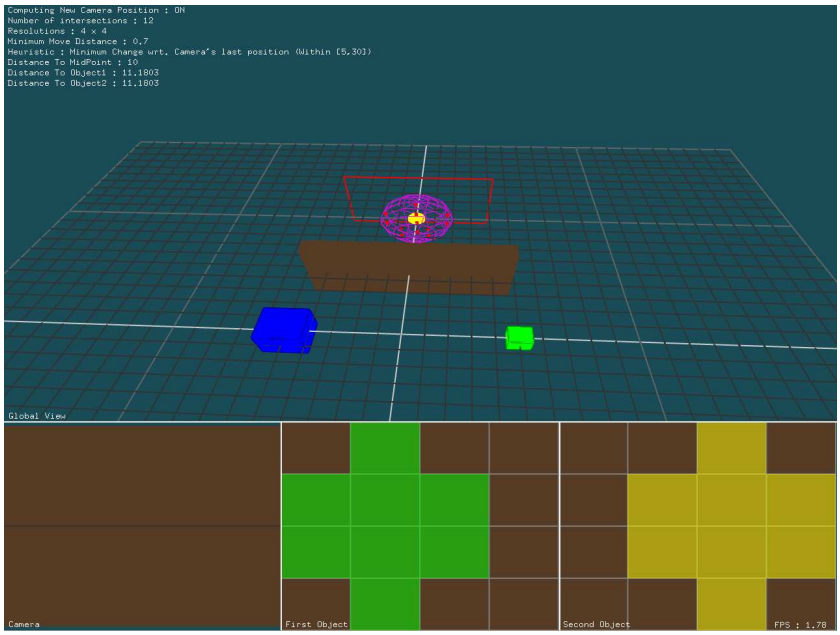



Figure 5.24 – Illustration d'une occultation des deux objets d'intérêt $O_1$ et $O_2$ dans le prototype OCCLUSION AVOIDER. Les points d'intersection appartenant à la sphère d'approximation sont affichés en rouge pour signaler que les deux objets d'intérêt sont occultés.

Cette heuristique de choix de point de vue, relative à la taille d'un polygone, est toutefois discutable. Il apparaît évident que, pour un modèle donné, la facette la plus représentée dans le *frame buffer* (*i.e.* celle dont l'aire est la plus grande) ne correspond pas nécessairement à une représentation équivoque de l'objet d'intérêt. Par exemple, lors de la modélisation d'un personnage en 3D, la tête de ce dernier correspond à une caractéristique particulièrement importante et sa modélisation consiste en un ensemble de polygones de taille réduite. Ainsi, alors que la tête d'un personnage est une caractéristique majeure de ce dernier, elle n'est pas considérée comme pertinente par l'heuristique de choix du polygone d'aire maximale. Afin de prendre en compte d'autres choix de points de projection en fonction des caractéristiques d'un modèle, il est possible de regrouper un ensemble de facettes au sein d'une notion de caractéristique sémantique, pour laquelle tous les polygones sont affichés avec la même couleur. Cela permet alors de mettre l'accent sur cette région du modèle 3D et ainsi de modifier le point de projection calculé. Toutefois, cette solution souffre des même limitations que la méthode du polygone d'aire maximale puisque le point retenu correspond au barycentre de toutes les facettes réunies au sein de la même « région sémantique » du modèle étudié. L'application MODELVIEWER qui permet de calculer les points de projection à choisir en fonction d'une position courante de caméra nécessite donc un travail important sur le choix des heuristiques permettant de déterminer les meilleurs points de projection associés à un objet d'intérêt.

Nous avons également incorporé la notion de vecteur d'orientation intrinsèque au sein de notre prototype MODELVIEWER, afin d'intégrer la notion cinématographique d'angle de vue à notre application d'évitement d'occlusion en environnements virtuels dynamiques : l'OCCLUSION AVOIDER. En effet, pour chacune des positions de caméra générées, nous pouvons déterminer l'angle de vue selon lequel la caméra filme l'objet d'intérêt. Ainsi, lors du choix du point de projection et en fonction de la position de caméra obtenue après évitement de l'occlusion pour les objets d'intérêt, nous sommes capables de calculer l'angle de vue entre l'objet d'intérêt et la caméra. Pour ce faire, étant donné une nouvelle position de caméra calculée, il suffit de parcourir la structure de données pour trouver la position de caméra la plus proche générée lors du prétraitement puis de récupérer l'angle de vue sous lequel cette position filme l'objet d'intérêt. Cela nous permet d'intégrer à la gestion de l'occlusion la notion d'angle de vue cinématographique. De plus, il est envisageable d'ajouter d'autres informations correspondant à différentes propriétés, cinématographiques ou non, afin de les incorporer dans le processus de traitement de l'occlusion. Nous pensons en particulier à la distance entre la caméra et l'objet qui permet de caractériser le type de plan de vue (gros plan, plan américain, etc.) selon lequel est filmé l'objet d'intérêt. Ainsi, nous pouvons aisément incorporer des notions cinématographiques au traitement des occlusions au sein d'une méthode interactive.

## 5.6 Conclusion

Nous avons proposé une approche nouvelle de l'évitement de l'occlusion en environnements temps réel dynamiques complexes. Notre approche est implémentée au sein d'un outil : l'OCCLUSION AVOIDER permettant de tester nos hypothèses de travail.

En introduction, nous avons identifié un certain nombre de problèmes liés à l'évitement de l'occlusion en environnement dynamique complexe. Nous les rappelons ici en précisant les réponses apportées par notre méthode :

- gestion d'un couple d'objets d'intérêt : nous définissons une méthode d'évitement de l'occlusion prenant en compte un couple d'objets, permettant ainsi d'en assurer la visibilité à l'écran. Comme nous l'avons précisé en introduction, la prise en compte de deux objets d'intérêt est intéressante

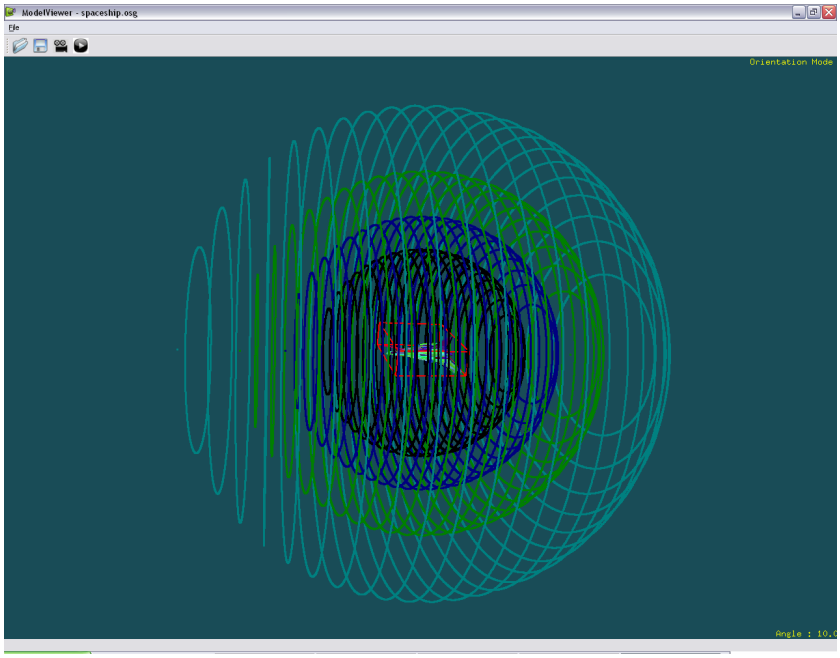



Figure 5.25 – Points de vue générés par le MODELVIEWER autour d'un modèle 3D à partir desquels une structure de données est créée afin d'ajouter des informations lors de l'évitement d'occlusions par l'application OCCLUSION AVOIDER.

pour bon nombre d'applications interactives comme les jeux vidéo ou les environnements d'apprentissage virtuels. En effet, cette spécificité assure la visibilité du personnage principal ou de l'avatar de l'étudiant et d'un objectif important lié à l'application.

- détection d'occlusion : nous proposons une détection précise et efficace des occlusions par utilisation de rendus matériel dans des tampons de profondeur, pouvant ainsi bénéficier d'améliorations de performances liées à l'évolution des cartes graphiques.
- réaction en cas d'occlusion : nous proposons une méthode permettant de déplacer la caméra vers des positions non occultées (dans la mesure du possible) en cas de détection d'occlusion pour la position courante de caméra. Lorsqu'aucune position de caméra non occultée n'existe, nous proposons un ensemble d'heuristiques permettant de déterminer une position satisfaisante en fonction d'un ensemble de critères définis par rapport au couple d'objets d'intérêt et à la caméra.
- gestion de l'aspect temporel : notre méthode prend en compte les événements précédents de l'environnement en accumulant au sein d'une structure de données les informations de profondeur précédentes. Ainsi, nous évitons de déplacer la caméra en réaction à des événements trop courts (occlusion d'objet très succincte) et assurons ainsi une cohérence à l'écran.

Toutefois, un certain nombre d'améliorations doivent encore être développées :

- implémentation : le prototype OCCLUSION AVOIDER a été développé sans réelle volonté d'optimisation du code. Ainsi, une augmentation des performances globales est aisément envisageable. En particulier en ayant recours aux accélérations matérielles proposées par OPENGL pour les rendus objets, par exemple l'extension *FBO* (*Frame Buffer Object*). De même, les calculs d'intersections des rayons peuvent bénéficier d'améliorations notables de performances grâce à l'utilisation de la programmation graphique (*GPU Programming*). Cette technique utilise le matériel graphique actuel afin d'effectuer efficacement les calculs en nombre flottants par le biais du processeur graphique (*GPU*) en lieu et place du processeur central (*CPU*) utilisé habituellement.
- extension à $n$ objets d'intérêt : notre approche est aisément extensible à $n$ objets d'intérêt, en ajoutant à chaque itération de l'algorithme :
  - deux rendus objets pour chaque couple d'objets pris en compte,
  - la fusion des informations de profondeur obtenues par les rendus précédents et stockées dans les repères non affines correspondants à chacun des couples d'objets.

Enfin, une occlusion au sein de l'OCCLUSION AVOIDER, est déterminée uniquement de façon géométrique c'est-à-dire en se basant sur les notions de recouvrement des projections d'objets à l'écran. Toutefois, une discussion est nécessaire concernant la notion même d'occlusion au sein d'une scène 3D. En effet, dans notre approche, comme dans les approches existantes d'évitement de l'occlusion, les notions de reconnaissance d'objets sont totalement délaissées, pourtant ces dernières apparaissent primordiales dans la gestion de l'occlusion. Nous devons considérer un objet comme occulté uniquement si celui-ci ne peut être reconnu ou identifié directement par l'utilisateur du système. Un exemple évident illustrant à quel point la caractérisation d'une occlusion est difficile peut être le déplacement d'un personnage dans un environnement virtuel représentant un paysage forestier. Si le personnage se déplace derrière une barrière, un grillage ou est filmé à travers les feuilles d'un arbre, doit-on le considérer comme occulté si celui-ci est reconnaissable?

La prise en compte de notions de haut niveau permet d'aboutir à une nouvelle définition « cognitive » de l'occlusion, à opposer à la définition géométrique utilisée actuellement. Toutefois, intégrer ces notions de reconnaissance nécessite une caractérisation sémantique des objets et une manière de détecter de quelle façon la reconnaissance d'un objet est effectuée par un humain. Un effort important est donc nécessaire dans les travaux futurs concernant les notions de reconnaissance, de caractérisation sémantique et de gestion de l'occlusion des objets en environnements virtuels.

# CHAPITRE 6

# Conclusion et perspectives

Nous avons présenté dans les chapitres précédents trois contributions à la problématique de placement de caméra en environnements virtuels. Ce chapitre dresse un bilan de chacune de nos contributions, présente leurs limitations et propose des pistes d'améliorations. Nous concluons ensuite ce manuscrit en proposant un ensemble de perspectives plus larges.

Le problème du contrôle d'une caméra virtuelle est complexe. En effet, lorsqu'un utilisateur de logiciel de modélisation 3D classique souhaite définir manuellement le placement d'une caméra virtuelle, celui-ci doit réaliser un processus d'*inversion mentale* qui lui permet d'inférer la position et l'orientation de la caméra en fonction de la vue qu'il souhaite obtenir de la scène 3D. Toutefois, il existe un ensemble de méthodes d'aide à l'utilisateur, qui offrent des propriétés permettant de s'abstraire d'un contrôle direct et bas niveau des paramètres d'une caméra virtuelle. Cependant, la majorité des approches de gestion de caméra en environnements virtuels sont spécifiques à une tâche (exploration proximale d'objets, navigation, suivi de cible, etc.) et peu adaptables en dehors des applications pour lesquelles elles ont été développées. *A contrario*, les méthodes « généralisées », principalement basées sur des méthodes d'optimisation et/ou de résolution de problèmes de satisfaction de contraintes, permettent l'étude de problèmes plus généraux, mais souffrent d'une insuffisance dans leur résolution de problèmes sur-contraints (*i.e.* dont la description initiale n'aboutit à aucune solution).

## 6.1 Une approche numérique pour le placement de caméra

Nous avons proposé au chapitre 3 un algorithme MAX-NCSP capable de résoudre indifféremment les problèmes sur-contraints, sous-contraints ou bien-contraints. L'algorithme proposé est générique et permet de résoudre tout type de problème de maximisation de satisfaction d'inéquations numériques (MAX-NCSP). En effet, nous proposons de calculer les pavés intérieurs maximisant le nombre de contraintes satisfaites d'un problème. L'étude se limite aux systèmes d'inéquations, c'est-à-dire pour lesquelles il est possible de calculer des régions intérieures. La propriété de *complétude* est assurée par une exploration de la totalité de l'espace de recherche et les zones satisfaisant de façon garantie le maximum des propriétés du problème sont calculées pour une précision donnée. Les solutions de l'algorithme vérifient la propriété de *correction* par rapport au nombre maximum $max$ de propriétés satisfaites, *i.e.* tous les points des boîtes solutions satisfont les contraintes associées aux $max$ propriétés certainement vérifiées. L'algorithme est basé sur une caractérisation $\mathcal{CPN}$ des boîtes de l'espace de recherche, correspondant respectivement aux ensembles de propriétés $\mathcal{C}$ertainement, $\mathcal{P}$ossiblement et $\mathcal{N}$on satisfaites pour toute boîte $\mathbf{B}$ de l'espace de recherche. Cette caractérisation nous permet d'éliminer rapidement les boîtes inintéressantes lors du processus de résolution. Une boîte de l'espace de recherche est éliminée dès

lors qu'elle ne peut satisfaire certainement un nombre de propriétés supérieur au maximum $m$ courant de l'algorithme MAX-NCSP.

Notre algorithme calcule l'ensemble des boîtes de l'espace de recherche maximisant le nombre de contraintes satisfaites pour un problème MAX-NCSP. Si le problème est bien-contraint ou sous-contraint, le nombre de contraintes satisfaites correspond au nombre de contraintes initial du problème. Si le problème est sur-contraint, le nombre de contraintes satisfaites est inférieur au nombre de contraintes du problème et chaque boîte solution est caractérisée par l'ensemble des contraintes certainement satisfaites. Notre algorithme est le seul, à notre connaissance, permettant la résolution de tout type de problème MAX-NCSP, indépendamment de la nature (sur, sous ou bien-contraint) de celui-ci.

Les méthodes classiques sont non seulement inadaptées à la prise en compte des problèmes sur-contraints, comme peuvent l'être les problèmes de placement de caméra, mais souffrent également de leur incapacité à fournir plusieurs solutions pour un problème. En effet, ces dernières sont principalement basées sur des méthodes d'optimisation numérique et/ou de satisfaction de contraintes (CSP) qui ne calculent qu'une solution unique pour un problème. Enfin, les méthodes existantes prenant en compte les problèmes sur-contraints, en particulier par relaxation de contraintes (*i.e.* retirer des contraintes du problème initial), fournissent une solution approchée au problème sans pour autant caractériser ni les contraintes satisfaites et non satisfaites du problème initial, ni les différentes classes de solution de ce dernier. Notre algorithme de résolution de problème MAX-NCSP permet une prise en compte native de problèmes sur, sous et bien-contraints tout en caractérisant les solutions calculées par l'ensemble de contraintes satisfaites.

Les performances de notre approche MAX-NCSP sont très dépendantes de la qualité du point de départ choisi, c'est pourquoi nous nous sommes intéressés aux méthodes de recherche locale, réputées pour leur capacité à calculer efficacement des solutions de CSPs discrets. Ainsi, nous avons développé une extension aux domaines continus d'une recherche locale avec métaheuristique taboue. Les concepts basiques inhérents à la recherche locale, à savoir les notions de configuration, de voisinage et d'évaluation ont été redéfinis pour une adaptation aux domaines continus et une implémentation a été proposée (cf. annexe C). Toutefois, l'extension aux domaines continus du cadre formel de la recherche locale que nous avons proposé consiste en un travail préliminaire auquel un certain nombre d'améliorations doivent être apportées. En particulier, une étude des différents opérateurs intervenant dans la résolution ainsi que de leur propriétés doit être menée afin d'évaluer leur impact sur l'algorithme.

La principale limitation de notre approche numérique de placement de caméra en environnements virtuels concerne la seule prise en compte d'inéquations. Le cadre tel que proposé ne peut pas être aisément étendu à des systèmes plus généraux, c'est-à-dire pas seulement composés d'inéquations numériques. Cette limitation est intrinsèque à notre approche, laquelle s'appuie sur les notions de continuums de solutions pour générer les ensembles maximisant le nombre de propriétés satisfaites.

## 6.2 Les volumes sémantiques

Afin de proposer une nouvelle méthode d'interaction avec l'utilisateur pour les problèmes de placement de caméra, nous avons introduit au chapitre 4 la notion de *volumes sémantiques* basée sur une caractérisation sémantique de l'espace de recherche et des solutions. Étant donné une description utilisateur correspondant à une liste de propriétés cinématographiques définies sur les objets d'une scène 3D, nous avons proposé une méthode en deux étapes :

1. une caractérisation de l'espace de recherche par partitionnement sémantique des positions de caméra en fonction d'un opérateur de filtrage géométrique ($G_f$),

2. un processus numérique calculant les meilleurs représentants de chacune des classes de solutions déterminées lors du filtrage géométrique.

La première étape consiste en l'application successive de l'opérateur $G_f$ à chacune des propriétés issues de la description utilisateur. Cet opérateur de filtrage géométrique calcule un *volume sémantique* englobant les positions de caméra permettant de satisfaire possiblement la propriété cinématographique à laquelle il est appliqué. Les volumes contenant les solutions du problème initial de placement de caméra sont ensuite calculés par intersection de tous les *volumes sémantiques*. Deux cas de figure se présentent :

- l'intersection est vide : la description du problème est inconsistante, *i.e.* celui-ci n'admet aucune solution. La méthode des volumes sémantiques permet d'avertir l'utilisateur du caractère sur-contraint de sa description sans avoir recours à des calculs numériques. Le processus géométrique permet à lui seul de détecter certaines inconsistances.
- l'intersection conduit à un ensemble de $n$ volumes non-connexes : le problème admet alors $n$ classes de solution distinctes correspondant à chacun des volumes. Chaque solution est sémantiquement équivalente, c'est-à-dire qu'elle satisfait l'ensemble de propriétés décrites par l'utilisateur, mais possède des caractéristiques visuelles spécifiques.

Une fois le nombre de classes de solutions déterminé, nous appliquons un processus numérique afin de calculer le « meilleur » représentant de chacun des volumes. Ces derniers contiennent uniquement les positions de caméra pouvant amener à la satisfaction d'un ensemble de propriétés cinématographiques, les orientations assurant le respect des contraintes restent encore à être calculées. Ceci est effectué par application de l'algorithme de recherche locale adapté aux domaines continus proposé dans le chapitre 3. Ce dernier semble particulièrement adapté du fait de la nature des contraintes numériques à traiter et de la nécessité d'efficacité relative au processus d'interaction avec l'utilisateur. À l'issue de ce processus numérique, nous sommes en mesure de proposer à l'utilisateur un représentant de chacun des volumes, *i.e.* de chaque classe de solution, maximisant la satisfaction des propriétés cinématographiques du problème de placement de caméra étudié. L'utilisateur peut alors choisir la solution qui lui convient le mieux en fonction des caractéristiques visuelles spécifiques à chaque classe de solution, ou bien choisir de raffiner le problème étudié en ajoutant des propriétés cinématographiques à sa description initiale.

Les volumes sémantiques proposent les fondements d'une nouvelle méthode d'interaction avec l'utilisateur pour les applications de contrôle de caméra en environnements virtuels. À l'inverse des approches existantes, nous souhaitons que, lorsqu'un problème de placement de caméra propose plusieurs classes de solutions équivalentes, la décision finale revienne à l'utilisateur, et non au processus de résolution. En effet, les approches classiques ne fournissent qu'une seule solution, c'est-à-dire une seule position et orientation de caméra virtuelle en réponse à une description utilisateur. *A contrario*, nous souhaitons proposer un exemple représentatif de chacune des classes de solutions ainsi que la liste des propriétés satisfaites par chacune d'elles.

La méthode des volumes sémantiques souffre toutefois d'un certain nombre de limitations. D'une part, les volumes sémantiques ne peuvent être appliqués qu'à des scènes 3D statiques. L'amélioration principale à apporter à notre méthode consiste en l'adaptation des volumes sémantiques aux environnements dynamiques. Cette mise en œuvre nécessite un algorithme de calcul de volumes temporels permettant de représenter l'évolution des différentes propriétés cinématographiques au cours du temps. Après avoir mené une étude concernant les algorithmes d'extrusion de volumes selon une trajectoire 3D et de création de volumes de balayage (*sweep volumes*), aucune solution ne nous semblait adaptée à une application interactive de création de volumes sémantiques temporels. Un effort certain permettant l'adaptation de cette nouvelle méthode d'interaction concerne donc le développement d'algorithmes efficaces de création de volumes 4D, évolution temporelle de volumes tridimensionnels.

D'autre part, un des objectifs des travaux menés sur les volumes sémantiques concerne leur incorporation dans les logiciels de modélisation 3D. Toutefois, cette interaction nécessite à la fois :

- le développement d'une interface utilisateur spécialisée dans l'exploitation des volumes et la présentation des résultats aux utilisateurs,
- l'amélioration des performances des algorithmes de création des volumes sémantiques.

L'exploitation et la présentation des volumes sémantiques aux utilisateurs nécessite la définition d'un langage expressif et interactif d'interrogation d'une scène 3D. En effet, un tel langage permettra l'enrichissement des propriétés existantes, la création d'opérateurs de composition et de structuration des résultats obtenus par les volumes sémantiques.

La création des volumes dans l'implémentation actuelle utilise une représentation par surfaces implicites facilitant la gestion des opérations Booléennes entre les volumes, mais nécessite une tesselation du résultat afin de déterminer le nombre de composantes connexes de l'intersection. Ce nombre correspond aux différentes classes de solutions sémantiquement équivalentes du problème. La tesselation est une opération très coûteuse d'un point de vue algorithmique. Une amélioration de la méthode actuelle consiste en l'implémentation efficace et robuste des opérations Booléennes entre volumes 3D. Ainsi, nous pourrions éviter la tesselation associée à l'utilisation des surfaces implicites pour améliorer l'interactivité avec les utilisateurs. Enfin, les interfaces d'interaction et d'exploitation des volumes sont envisageables à la manière de greffons spécialisés (*plug-ins*) permettant ainsi d'intégrer aisément la méthode des volumes sémantiques dans les logiciels existants.

## 6.3 La gestion de l'occlusion

Nous avons mis l'accent, tout au long de ce manuscrit, sur l'importance que revêt la prise en compte de la notion d'occlusion dans le contrôle de caméra. Nous avons développé un prototype illustrant une nouvelle approche : l'OCCLUSION AVOIDER. Nous proposons de gérer l'occlusion simultanée de deux objets d'intérêt dans un environnement temps réel dynamique. Notre méthode est basée sur l'utilisation de rendus matériels dans des tampons de profondeur (*depth buffers*) depuis les objets d'intérêt vers la position courante de caméra. Ceci permet de calculer de manière précise et efficace un ensemble de positions potentielles de caméra pour lesquelles le statut d'occlusion, par rapport aux objets d'intérêt, est déterminé. Nous proposons également un ensemble d'heuristiques permettant de discriminer les positions équivalentes en termes d'occlusion par rapport aux objets et proposons une évaluation de ces dernières sur un ensemble de jeux de test. L'OCCLUSION AVOIDER permet en outre la prise en compte d'une dimension temporelle lors de la gestion de l'occlusion en accumulant les informations précédentes d'occlusion pour les positions de caméra. La prise en compte de cette notion temporelle représente une « mémoire » des occlusions précédentes. Elle permet d'éviter de réagir à des événements ponctuels et ainsi de préserver une cohérence à l'image en évitant les sauts de caméra.

Notre méthode de gestion de l'occlusion en environnement dynamique temps réel est extensible à un nombre arbitraire d'objets lorsque ceux-ci sont pris en compte deux par deux. Toutefois, un travail d'optimisation du code est nécessaire pour que cette gestion multi-objets puisse être effectuée en temps réel. Un certain nombre d'améliorations peuvent d'ores et déjà être implémentées dans l'OCCLUSION AVOIDER, en particulier concernant les rendus objets effectués dans les tampons de profondeur. Ceux-ci peuvent en effet bénéficier des améliorations du matériel graphique actuel, par exemple en utilisant les extensions *Frame Buffer Objects* (*FBO*) d'OPENGL. De même, la majorité des opérations mathématiques peuvent profiter des avantages de la programmation graphique (*GPU Programming*) accélérant ainsi notablement les calculs réalisés.

## 6.4 Conclusion

Dans cette thèse, nous avons cherché à surmonter un certain nombre de limitations des approches existantes à la prise en compte du problème de placement de caméra en environnement virtuel. Nous avons proposé une méthode offrant une plus grande liberté à l'utilisateur en termes de résolution d'un problème. Notre méthode prend en compte indifféremment les problèmes sur-contraints, sous-contraints ou bien-contraints, et permet ainsi une plus grande liberté à l'utilisateur e termes de spécification et de résolution d'un problème de placement de caméra.

La méthode des volumes sémantiques se démarque des approches existantes en présentant à l'utilisateur l'ensemble des solutions équivalentes en termes de satisfaction de propriétés d'un problème de placement de caméra. Cette méthode est basée sur la définition d'un nouveau mode d'interaction avec l'utilisateur permettant à celui-ci d'intérroger et d'interagir avec les potentialités cinématographiques d'une scène 3D.

Enfin, une méthode précise et efficace de gestion de l'occlusion simultanée de plusieurs objets d'intérêt a été proposée. Cette dernière peut être étendue afin de prendre également en compte des propriétés cinématographiques (*e.g.* orientation à l'écran, plan de vue d'un objet, etc.).

Toutefois, le problème de placement de caméra en environnement virtuel est encore loin d'être totalement résolu. En particulier, toutes les approches existantes, tout comme les solutions proposées dans cette thèse prennent uniquement en compte les aspects géométriques des objets, des caméras virtuelles et des propriétés (cinématographiques ou non). Nous sommes convaincus que les techniques doivent s'abstraire du contrôle bas niveau de la caméra et des objets 3D et délaisser les aspects géométriques pour se concentrer sur les niveaux sémantiques, cognitifs et esthétiques empruntés aux domaines de la photographie, du cinéma et de la psychologie cognitive.

La nécessité de modèles sémantiques liés aux objets permet une augmentation de l'expressivité associée à la spécification de propriétés, en incorporant les aspects suivants :

- caractérisation sémantique des différentes « parties » d'un objet et de leurs aspects fonctionnels,
- orientation intrinsèque,
- définition de relations spatiales dans la scène,

Un tel modèle est à prendre en compte dans l'optique de pouvoir spécifier des propriétés de haut niveau à la fois dans l'espace image et dans la scène 3D. De même, la spécification des propriétés en termes esthétiques plutôt qu'en fonction de critères uniquement géométriques faciliterait l'analogie possible entre la gestion de placement de caméra en environnement virtuel et les techniques employées en cinéma ou photographie. Ainsi, une prise en compte des notions de montage, de lumière, de balance visuelle, d'ombrage ou de lignes de fuite permettrait une expressivité beaucoup plus importante pour les utilisateurs et augmenterait significativement la qualité des résultats produits pour les modélisations de trajectoires ou de placement de caméras virtuelles. De même, la propriété d'occlusion nécessite la prise en compte de modèles cognitifs de reconnaissance d'objets et de compréhension temporelle d'une scène 3D. Nous pensons que les recherches futures doivent être dirigées vers la prise en compte de ces propriétés de haut niveau, tant concernant la représentation des objets et de la caméra, que l'expressivité alliée aux propriétés utilisateur.

Enfin, la notion de montage audiovisuel, qui consiste à agencer différents plans entre eux afin de former des séquences cohérentes ou des transitions entre plusieurs séquences, est pour le moment absente des approches de contrôle de caméra en environnements virtuels. Néanmoins, la prise en compte de cet aspect augmenterait significativement le pouvoir expressif des utilisateurs de systèmes de contrôle de caméra, en offrant aux réalisateurs virtuels les mêmes possibilités que celles de la cinématographie classique.

Ainsi, afin d'élever l'impact des images virtuelles (*i.e.*3D) au même niveau que celui des images réelles, il est nécessaire de travailler à la fois sur des aspects techniques fondamentaux (bli), mais également, dans une approche trans-disciplinaire (pluri?) de s'intéresser à la perception, à l'analyse et à la compréhension des fondements sémantiques, cognitifs et esthétiques de l'image par l'Homme, dans le but de retranscrire ces mécanismes lors de la production d'images virtuelles.

# Bibliographie


[ACZ04] ANGLADA A., CODOGNET P., ZIMMER L.: An adaptive search for the NSCSPs. In *Proceedings of the 8th ERCIM/CoLogNet workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2004)* (Lausanne, 2004).

[AH83] ALEFELD G., HERZBERGER J.: *Introduction to Interval Computations*. Academic Press Inc., New York, USA, 1983.

[AMH02] AKENINE-MÖLLER T., HAINES E.: *Real-Time Rendering*, 2nd ed. A. K. Peters, Ltd., Natick, MA, USA, 2002.

[Ari76] ARIJON D.: *Grammar of the Film Language*. Hastings House Publishers, 1976.

[Aum90] AUMONT J.: *L'image*, 2 ed. Nathan, 1990.

[AVF04] ANDÚJAR C. G., VÁZQUEZ P. P. A., FAIRÉN M. G.: Way-finder: Guided tours through complex walkthrough models. *Comput. Graph. Forum 23*, 3 (2004), 499–508.

[Bau72] BAUMGART B. G.: *Winged edge polyhedron representation.* Tech. rep., Stanford University, Stanford, CA, USA, 1972.

[BCDP07] BELDICEANU N., CARLSSON M., DEMASSEY S., PETIT T.: Global constraint catalogue: Past, present and future. *Constraints 12*, 1 (2007), 21–62.

[Bec02] BECKHAUS S.: *Dynamic Potential Fields for Guided Exlporation in Virtual Environments*. PhD thesis, Fakultät für Informatik, University of Magdeburg, 2002.

[Ben95] BENHAMOU F.: Interval constraint logic programming. In *Constraint Programming: Basics and Trends, LNCS no 910*, Podelski A., (Ed.). Springer Verlag, 1995, pp. 1–21.

[Bes94] BESSIÈRE C.: Arc-consistency and arc-consistency again. *Artificial Intelligence 65*, 1 (1994), 179–190.

[BG00] BENHAMOU F., GOUALARD F.: Universally quantified interval constraints. In *Proceedings of International Conference on Principles and Practice of Constraint Programming* (Singapore, 2000), Dechter R., (Ed.), vol. 1894 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 67–82.

[BGL98] BARES W. H., GREGOIRE J. P., LESTER J. C.: Realtime Constraint-Based Cinematography for Complex Interactive 3D Worlds. In *Procs of AAAI-98/IAAI-98* (1998), pp. 1101–1106.

[BH03] BARICHARD V., HAO J.-K.: A population and interval constraint propagation algorithm. *LNCS 2632* (2003), 88–101.

[BKF*02] BURTNYK R., KHAN A., FITZMAURICE G., BALAKRISHAN R., KURTENBACH G.: Stylecam: Interactive stylized 3d navigation using integrated spatial & temporal controls. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2002), ACM, pp. 101–110.

[BKFK06] BURTNYK N., KHAN A., FITZMAURICE G., KURTENBACH G.: Showmotion: camera motion based 3d design review. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), ACM Press, pp. 167–174.

[BL97] BARES W. H., LESTER J. C.: Cinematographic user models for automated realtime camera control in dynamic 3D environments. In *Proceedings of the sixth International Conference on User Modeling* (Vien New York, 1997), Jameson A Paris C T. C., (Ed.), Springer-Verlag, pp. 215–226.

[BL99] BARES W. H., LESTER J. C.: Intelligent Multi-Shot Visualization Interfaces for Dynamic 3D Worlds. In *IUI '99: Proceedings of the 4th international conference on Intelligent user interfaces* (New York, NY, USA, 1999), ACM Press, pp. 119–126.

[Bli88] BLINN J.: Where am I? what am I looking at? *IEEE Computer Graphics and Applications* (July 1988), 76–81.

[BMBT00] BARES W., MCDERMOTT S., BOUDREAUX C., THAINIMIT S.: Virtual 3D Camera Composition from Frame Constraints. In *MULTIMEDIA '00: Proceedings. of the eighth ACM international conference on Multimedia* (New York, NY, USA, 2000), ACM Press, pp. 177–186.

[BMvH94] BENHAMOU F., MCALLESTER D., VAN HENTENRYCK P.: Clp(intervals) revisited. In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming* (Cambridge, MA, USA, 1994), MIT Press, pp. 124–138.

[BO02] BRADSHAW G., O'SULLIVAN C.: Sphere-tree construction using dynamic medial axis approximation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), ACM Press, pp. 33–40.

[BR03] BLUM C., ROLI A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys 35* (2003), 268–308.

[Bra75] BRAID I. C.: The synthesis of solids bounded by many faces. *Commun. ACM 18*, 4 (1975), 209–216.

[BRS01] BECKHAUS S., RITTER F., STROTHOTTE T.: Guided exploration with dynamic potential fields: the cubicalpath system. *Comput. Graph. Forum 20*, 4 (2001), 201–210.

[BRZL98] BARES W. H., RODRIGUEZ D. W., ZETTLEMOYER L. S., LESTER J. C.: Task-Sensitive Cinematography Interfaces for Interactive 3D Learning Environments. In *Proceedings Fourth International conference on Intelligent User Interfaces (IUI'98)* (1998), pp. 81–88.

[BT96] BATTITI R., TECCHIOLLI G.: The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization. *Annals of Operations Research 62* (1996).

[But97] BUTZ A.: Animation with CATHI. In *Proceedings of American Association for Artificial Intelligence/AAAI '97* (1997), AAAI Press, pp. 957–962.

[CAH*96] CHRISTIANSON D. B., ANDERSON S. E., HE L., SALESIN D. H., WELD D. S., COHEN M. F.: Declarative Camera Control for Automatic Cinematography. In *Proceedings of the American Association for Artificial Intelligence/ AAAI '96* (1996), AAAI Press, pp. 148–155.

[CDR99] COLLAVIZZA H., DELOBEL F., RUEHER M.: Extending consistent domains of numeric CSP. In *Procs. of the 16th IJCAI* (Stockholm, Sweden, July 1999), vol. 1, pp. 406–411.

[CG85] COHEN M. F., GREENBERG D. P.: The hemi-cube: a radiosity solution for complex environments. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1985), ACM Press, pp. 31–40.

[Chr00] CHRISTIE M.: Universal solution viewer. In *Proceedings SCAN/INTERVAL 2000* (Sept. 2000), IMACS.

[Chr03] CHRISTIE M.: *Spécification de trajectoires de caméra sous contraintes*. PhD thesis, Université de Nantes, 2003.

[CL03] CHRISTIE M., LANGUÉNOU E.: A Constraint-Based Approach to Camera Path Planning. In *Proceedings of the Third International Symposium on Smart Graphics* (2003), vol. 2733 of *Lecture Notes in Computer Science*, Springer, pp. 172–181.

[CLDM03] COURTY N., LAMARCHE F., DONIKIAN S., MARCHAND E.: A Cinematography System for Virtual Storytelling. In *Int. Conf. on Virtual Storytelling, ICVS'03* (Toulouse, France, November 2003), Balet O., Subsol G., Torguet P., (Eds.), vol. 2897 of *Lecture Notes in Computer Science*, pp. 30–34.

[CLG02] CHRISTIE M., LANGUÉNOU E., GRANVILLIERS L.: Modeling Camera Control with Constrained Hypertubes. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming* (Ithaca, NY, USA, September 9-13 2002), Lecture Notes in Computer Science (LNCS), Springer-Verlag, pp. 618–632.

[CM01] COURTY N., MARCHAND E.: Computer animation: A new application for image-based visual servoing. In *In Proceedings of IEEE Int. Conf. on Robotics and Automation, ICRA'2001* (2001), vol. 1, pp. 223–228.

[CMS88] CHEN M., MOUNTFORD S., SELLEN A.: A study in interactive 3d rotation using 2d input devices. In *Computer Graphics (Proceedings SIGGRAPH '88)* (Aug. 1988), Catmull E. E., (Ed.), vol. 22-4, pp. 121–130.

[CN05] CHRISTIE M., NORMAND J.-M.: A semantic space partitioning approach to virtual camera composition. *Computer Graphics Forum, Eurographics 2005 conference proceedings 24*, 3 (2005), 247–256.

[Cou02] COURTY N.: *Animation référencée vision : de la tâche au comportement*. PhD thesis, INSA Rennes, soutenue à l'Université de Rennes I, November 2002.

[CS00] CHELOUAH R., SIARRY P.: Tabu search applied to global optimization. *European Journal on Operational Research* (2000).

[CS03] CHELOUAH R., SIARRY P.: Genetic and nelder-mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions. *European Journal of Operational Research 148* (2003), 335–348.

[DDP02] DURAND F., DRETTAKIS G., PUECH C.: The 3D Visibility Complex. *ACM Transactions on Graphics 21*, 2 (April 2002), 176–206.

[DGZ92] DRUCKER S. M., GALYEAN T. A., ZELTZER D.: Cinema: A System for Procedural Camera Movements. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics* (New York, NY, USA, 1992), ACM Press, pp. 67–70.

[DN03] DENNIS NIEUWENHUISEN M. H. O.: *Motion Planning for Camera Movements in Virtual Environments*. Tech. Rep. UU-CS-2003-004, Institute of Information and Computing Sciences, Utrecht University, 2003.

[Dru94] DRUCKER S. M.: *Intelligent Camera Control for Graphical Environments*. PhD thesis, School of Architecture and Planning, Massachusetts Institute of Technology MIT Media Lab, 1994.

[dS16] DE SAUSSURE F.: *Cours de linguistique générale*. Bayot, 1916.

[DZ95] DRUCKER S. M., ZELTZER D.: Camdroid: a system for implementing intelligent camera control. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics* (New York, NY, USA, 1995), ACM Press, pp. 139–144.

[ECR92] ESPIAU B., CHAUMETTE F., RIVES P.: A new approach to visual servoing in robotics. *IEEE Trans. on Robotics and Automation 8*, 3 (June 1992), 313–326.

[Fec60] FECHNER G. T.: *Elemente der Psychophysik*. Breitkof und Hartel, 1860.

[FKN79] FUCHS H., KEDEM Z. M., NAYLOR B. F.: Predetermining visibility priority in 3-d scenes (preliminary report). In *SIGGRAPH '79: Proceedings of the 6th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1979), ACM Press, pp. 175–181.

[FKN80] FUCHS H., KEDEM Z. M., NAYLOR B. F.: On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1980), ACM Press, pp. 124–133.

[FR89] FEO T., RESENDE M.: A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters 8* (1989), 67–71.

[FR95] FEO T., RESENDE M.: Greedy randomized adaptive search procedures. *Journal of Global Optimization 6* (1995), 109–133.

[FvDFH90] FOLEY J. D., VAN DAM A., FEINER S. K., HUGHES J. F.: *Computer Graphics: Principles and Practice*, 2nd ed. Addison-Wesley Publishing Co., Reading, MA, 1990.

[Gio04] GIORS J.: The full spectrum warrior camera system. In *GDC '04 : Game Developers Conference 2004* (2004).

[GL97] GLOVER F., LAGUNA M.: *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.

[Glo86] GLOVER F.: Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research 13*, 5 (1986), 533–549.

[Gol91] GOLDBERG D.: What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv. 23*, 1 (1991), 5–48.

[Gou00] GOUALARD F.: *Langages et environnements en programmation par contraintes d'intervalles*. PhD thesis, IRIN, Université de Nantes, 2000.

[GPZT98] GOBBETTI E., PILI P., ZORCOLO A., TUVERI M.: Interactive virtual angioscopy. In *VIS '98: Proceedings of the conference on Visualization '98* (Los Alamitos, CA, USA, 1998), IEEE Computer Society Press, pp. 435–438.

[GRMS01] GOOCH B., REINHARD E., MOULDING C., SHIRLEY P.: Artistic composition for image creation. In *Eurographics Workshop on Rendering* (2001), pp. 83–88.

[GW92] GLEICHER M., WITKIN A.: Through-the-lens camera control. In *Proceedings of ACM SIGGRAPH'92* (1992), pp. 331–340.

[Han86] HANSEN P.: The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization* (1986).

[HCS96] HE L., COHEN M. F., SALESIN D. H.: The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *SIGGRAPH 96 Conference Proceedings* (August 1996), Rushmeier H., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 217–224. held in New Orleans, Louisiana, 04-09 August 1996.

[HGH98] HAO J.-K., GALINIER P., HABIB M.: Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Journal of Heuristics* (1998).

[HHO05] HOWLETT S., HAMILL J., O'SULLIVAN C.: Predicting and evaluating saliency for simplified polygonal models. *ACM Trans. Appl. Percept. 2*, 3 (y 05), 286–308.

[HHS01] HALPER N., HELBING R., STROTHOTTE T.: A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. In *Proceedings of the Eurographics'2001 Conference* (2001), vol. 20, pp. 174–183.

[HMK*97] HONG L., MURAKI S., KAUFMAN A., BARTZ D., HE T.: Virtual voyage: interactive navigation in the human colon. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 27–34.

[HMPR07] HIRSCH M., MENESES C., PARDALOS P., RESENDE M.: Global optimization by continuous GRASP. *Optimization Letters 1* (2007), 201–212.

[HO00] HALPER N., OLIVIER P.: CAMPLAN: A Camera Planning Agent. In *Smart Graphics 2000 AAAI Spring Symposium* (March 2000), pp. 92–100.

[Hof89] HOFFMANN C. M.: *Geometric and solid modeling: an introduction*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[HW97] HANSON A., WERNERT E.: Constrained 3d navigation with 2d controllers. In *IEEE Visualization* (1997), pp. 175–182.

[IEE85] IEEE: *IEEE Standard for Binary Floating-Point Arithmetic*. Tech. Rep. IEEE Std 754-1985, Institute of Electrical and Electronics Engineers, 1985. Reaffirmed 1990.

[JFM96] JAMPEL M., FREUDER E. C., MAHER M. J. (Eds.):. *Over-Constrained Systems* (1996), vol. 1106 of *Lecture Notes in Computer Science*, Springer.

[JL98] JARDILLIER F., LANGUÉNOU E.: Screen-Space Constraints for Camera Movements: the Virtual Cameraman. In *Proceedings of the Eurographics'98 Conference* (1998), Ferreira N., Göbel M., (Eds.), vol. 17, Blackwell Publishers, pp. 175–186. ISSN 1067-7055.

[JLS*97] JOLESZ F., LORENSEN W., SHINMOTO H., ATSUMI H., NAKAJIMA S., KAVANAUGH P., SAIVIROONPORN P., SELTZER S., SILVERMAN S., PHILLIPS M., KIKINIS R.: Interactive virtual endoscopy. *American Journal of Roentgenology 169*, 5 (November 1997), 1229–1235.

[Joh03] JOHNSTON M.: Multimodality in language and speech systems edited by björn granström, david house, and inger karlsson. *Computational Linguistics 29*, 2 (2003), 321–324.

[Jus06] JUSSIEN N.: *Précis de Sudoku*. Hermes Science, 2006.

[JW93] JAULIN L., WALTER E.: Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica 29*, 4 (1993), 1053–1064.

[Kat91] KATZ S.: *Film Directing Shot by Shot: Visualizing from Concept to Screen*. Michael Wiese Productions, 1991.

[KKH95] KUNG M. H., KIM M. S., HONG S.: Through-the-lens camera control with a simple jacobian matrix. In *Proceedings of Graphics Interface '95* (1995), pp. 117–178.

[KKS*05] KHAN A., KOMALO B., STAM J., FITZMAURICE G., KURTENBACH G.: Hovercam: interactive 3d navigation for proximal object inspection. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM Press, pp. 73–80.

[Knu81] KNUTH D. E.: *Seminumerical Algorithms*, second ed., vol. 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, 10 Jan. 1981.

[KvD79] KOENDERINK J., VAN DOORN J.: The internal representation of solid shape with respect to vision. *Biological Cybernetics 32* (1979), 211–216.

[Lat91] LATOMBE J.: *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[LBGC98] LANGUÉNOU E., BENHAMOU F., GOUALARD F., CHRISTIE M.: The virtual cameraman: an interval constraint based approach. In *Proceedings of the ECAI'98 workshop Constraint techniques for artistic applications* (1998).

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM Press, pp. 163–169.

[LM01] LEFÉVRE V., MULLER J.-M.: Worst cases for correct rounding of the elementary functions in double precision. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic* (Vail, Colorado, USA, 2001), Burgess N., Ciminiera L., (Eds.), pp. 111–118.

[LMT98] LEFÈVRE V., MULLER J.-M., TISSERAND A.: Toward correctly rounded transcendentals. *IEEE Trans. Comput. 47*, 11 (1998), 1235–1243.

[LT00] LI T.-Y., TING H.-K.: An intelligent user interface with motion planning for 3d navigation. In *Proceedings of the IEEE Virtual Reality 2000 Conference* (2000), pp. 177–184.

[LVJ05] LEE C. H., VARSHNEY A., JACOBS D. W.: Mesh saliency. *ACM Trans. Graph. 24*, 3 (2005), 659–666.

[Mac77] MACKWORTH A. K.: Consistency in networks of relations. *Artifical Intelligence 8*, 1 (1977), 99–118.

[Mas65] MASCELLI J.: *The Five C's of Cinematography: Motion Picture Filming Techniques*. Cine/Grafic Publications, Hollywood, 1965.

[MC00] MARCHAND E., COURTY N.: Image-based virtual camera motion strategies. In *Proc. of Graphics Interface, GI2000* (Montréal, May 2000), pp. 69–76.

[MC02] MARCHAND E., COURTY N.: Controlling a camera in a virtual environment. *The Visual Computer Journal 18*, 1 (2002), 1–19.

[MH97] MLADENOVIĆ N., HANSEN P.: Variable neighborhood search. *Computers & OR 24*, 11 (1997), 1097–1100.

[MH98] MARCHAND E., HAGER G.: Dynamic sensor planning in visual servoing. In *IEEE Int. Conf. on Robotics and Automation, ICRA'98* (Leuven, Belgium, May 1998), vol. 3, pp. 1988–1993.

[Mic95] MICHALEWICZ Z.: Genetic Algorithms, Numerical Optimization and Constraints. *Proceedings of the 6th International Conference on Genetic Algorithms* (1995), 151–158.

[MJPL92] MINTON S., JOHNSTON M., PHILIPS A., LAIRD P.: Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence 58* (1992), 161–205.

[MLB02] MCDERMOTT S., LI J., BARES W.: Storyboard Frame Editing for Cinematic Composition. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces* (New York, NY, USA, 2002), ACM Press, pp. 206–207.

[Moo66] MOORE R. E.: *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[MvH02] MICHEL L., VAN HENTENRYCK P.: A constraint-based architecture for local search. *SIGPLAN Not. 37*, 11 (2002), 83–100.

[Neu90] NEUMAIER A.: *Interval methods for systems of equations*, vol. 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge Unversity Press, 1990.

[NLK93] NODINEM C., LOCHER J., KRUNPINSKI E.: *The role of formal art training on perception and aesthetic judgement of art compoistions*. Leonardo, 1993.

[OHPL99] OLIVIER P., HALPER N., PICKERING J., LUNA P.: Visual Composition as Optimisation. In *AISB Symposium on AI and Creativity in Entertainment and Visual Art* (1999), pp. 22–30.

[Pal96] PALAMIDESE P.: A Camera Motion Metaphor Based on Film Grammar. *Journal of Visualization and Computer Animation 7*, 2 (1996), 61–78.

[PBG92] PHILLIPS C. B., BADLER N. I., GRANIERI J.: Automatic viewing control for 3d direct manipulation. In *Proceedings of the 1992 symposium on Interactive 3D graphics* (1992), ACM Press New York, NY, USA, pp. 71–74.

[PD90] PLANTINGA H., DYER C. R.: Visibility, Occlusion, and the aspect graph. *International Journal of Computer Vision 5*, 2 (Nov 1990), 137–160.

[Pei78] PEIRCE C. S.: *Écrits sur le signe*. Seuil, 1978.

[Pic02] PICKERING J. H.: *Intelligent Camera Planning for Computer Graphics*. PhD thesis, Department of Computer Science, University of York, September 2002.

[Rum88] RUMP S. M.: Algorithms for verified inclusions — theory and practice. In *Reliability in Computing, Perspectives in Computing* (1988), Moore R. E., (Ed.), Academic Press, pp. 109–126.

[SF91] SELIGMANN D. D., FEINER S.: Automated generation of intent-based 3d illustrations. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1991), ACM Press, pp. 123–132.

[SF93] SELIGMANN D. D., FEINER S.: Supporting interactivity in automated 3d illustrations. In *IUI '93: Proceedings of the 1st international conference on Intelligent user interfaces* (New York, NY, USA, 1993), ACM Press, pp. 37–44.

[SGLM03] SALOMON B., GARBER M., LIN M. C., MANOCHA D.: Interactive navigation in complex environments using path planning. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics* (New York, NY, USA, 2003), ACM Press, pp. 41–50.

[Sha48] SHANNON C. E.: A mathematical theory of communication. *The Bell System technical journal 27* (1948), 379–423.

[SHF94] SAM-HAROUD D., FALTINGS B.: Global consistency for continuous constraints. In *PPCP '94: Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming* (London, UK, 1994), Springer-Verlag, pp. 40–50.

[Sho92] SHOEMAKE K.: Arcball: a user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of Graphics Interface '92* (May 1992), pp. 151–156.

[SML98] SCHROEDER W., MARTIN K. M., LORENSEN W. E.: *The visualization toolkit (2nd ed.): an object-oriented approach to 3D graphics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[Sny92] SNYDER J.: Interval analysis for computer graphics. In *J. M. Snyder, Interval analysis for computer graphics. Proceedings of SIGGRAPH'92, in ACM Computer Graphics 26, 2 (July 1992), 121–130.* (1992).

[SWND05] SHREINER D., WOO M., NEIDER J., DAVIS T.: *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2 (5th Edition) (OpenGL)*. Addison-Wesley Professional, Boston, MA, USA, 2005.

[TBGT91] TURNER R., BALAGUER F., GOBBETTI E., THALMANN D.: Physically-based interactive camera motion control using 3D input devices. In *Scientific Visualization of Physical Phenomena*, Patrikalakis N. M., (Ed.). Springer Verlag, 1991, pp. 135–145.

[TBN00] TOMLINSON B., BLUMBERG B., NAIN D.: Expressive autonomous cinematography for interactive virtual environments. In *Proceedings of the Fourth International Conference on Autonomous Agents* (Barcelona, Catalonia, Spain, 2000), Sierra C., Gini M., Rosenschein J. S., (Eds.), ACM Press, pp. 317–324.

[Tis95] TISSERON S.: *Psychanalyse de l'image – Des premiers traits au virtuel*, 2 ed. Dunod, 1995.

[Tis96] TISSERON S.: *Le bonheur dans l'image*. Les empêcheurs de penser en rond, 1996.

[Tsa93] TSANG E. P.: *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[vH67] VON HELMHOLTZ H.: *Handbuch der physiologischen Optik*. Leipzig, Voss, 1867.

[VMD97] VAN HENTENRYCK P., MICHEL L., DEVILLE Y.: *Numerica: A Modeling Language for Global Optimization*. The MIT Press, 1997.

[VSHS02] VU X.-H., SAM-HAROUD D., SILAGHI M.-C.: Approximation techniques for non-linear problems with continuum of solutions. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation* (London, UK, 2002), Springer-Verlag, pp. 224–241.

[WO90] WARE C., OSBORNE S.: Exploration and virtual camera control in virtual three dimensional environments. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics* (New York, NY, USA, 1990), ACM Press, pp. 175–183.

[WW89] WYVILL B., WYVILL G.: Field functions for implicit surfaces. *The Visual Computer 5*, 1&2 (1989), 75–82.

[XH98] XIAO D., HUBBOLD R. J.: Navigation guided by artificial force fields. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems* (1998), Wesley A., (Ed.), pp. 179–186.

[ZF99] ZELEZNIK R. C., FORSBERG A. S.: Unicam - 2d gestural camera controls for 3d environments. In *Symposium on Interactive 3D Graphics* (1999), pp. 169–173.

[ZRS03] ZANCANARO M., ROCCHI C., STOCK O.: Automatic video composition. In *Proceedings of the Third International Symposium on Smart Graphics* (2003), vol. 2733 of *Lecture Notes in Computer Science*, Springer, pp. 192 – 201.

[ZSA03] ZANCANARO M., STOCK O., ALFARO I.: Using cinematic techniques in a multimedia museum guide. In *Proceedings of Museums and the Web 2003* (March 2003).

# Références hypertextes


[www1] *The Visualization Toolkit.*
http://www.vtk.org/

[www2] *The Tool Command Language (Tcl) and the Tk graphical user interface toolkit.*
http://www.tcl.tk/

[www3] *Swept Volume Computation Reading List.*
http://www.cs.unc.edu/~youngkim/sweep/

[www4] *The GNU Triangulated Surface Library.*
http://gts.sourceforge.net/

[www5] *Échelle des plans de vue sur la Wikipedia.*
http://fr.wikipedia.org/wiki/Cadre_(art)

[www6] *Full Spectrum Warrior.*
http://www.fullspectrumwarrior.com/fr/

[www7] *idSoftware, société de développement de jeux vidéo comme Doom ou Quake.*
http://www.idsoftware.com

[www8] *Décomposition en valeurs singulières sur la wikipédia.*
http://fr.wikipedia.org/wiki/Décomposition_en_valeurs_singulières

[www9] *L'électromagnétisme sur la Wikipedia.*
http://fr.wikipedia.org/wiki/Electromagnétisme

[www10] *Catalogue des contraintes globales.*
http://www.emn.fr/x-info/sdemasse/gccat/index.html

[www11] *Plan séquence sur la wikipédia.*
http://fr.wikipedia.org/wiki/Plan_séquence

[www13] *Guide à la programmation par contraintes.*
http://kti.mff.cuni.cz/~bartak/constraints/index.html

[WWW14] *Théorie de la complexité sur la wikipédia.*
http://fr.wikipedia.org/wiki/Classes_de_complexité_P_et_NP

[WWW15] *Définition du maillage sur la Wikipédia.*
http://fr.wikipedia.org/wiki/Maillage

[WWW17] *Logiciel de modélisation Softimage XSI 3D.*
http://www.softimage.com/products/xsi/

[WWW18] *La sémiotique sur la Wikipédia.*
http://fr.wikipedia.org/wiki/Sémiotique

# Liste des tableaux

# Table des figures

# Table des exemples

# Table des matières

# Annexes

# ANNEXE A

# Modèles de représentation

Cette section est consacrée à la présentation des modèles de représentation des objets et de la caméra qui sont utilisées tout au long de cette thèse. Sauf indication contraire, lorsque nous ferons référence aux paramètres de la caméra ou d'un objet, la représentation sous-jacente sera celle présentée ici. Nous présentons tout d'abord le modèle de caméra, avant de nous intéresser à celui des objets.

### A.0.1 Modèle de représentation de caméra

Le modèle de caméra que nous utilisons est basé sur la définition de sept paramètres pour une caméra $C$ :

- trois coordonnées $(X_C, Y_C, Z_C)$ définissant la position de la caméra dans la scène,
- trois angles $(\phi_C, \theta_C, \psi_C)$ correspondants aux angles d'Euler, définissant l'orientation de la caméra et représentant les rotations respectives autour des axes $X, Y, Z$ du repère global,
- une distance focale $\gamma_C$ représentant l'ouverture focale de la caméra.

Les sept paramètres définissant une caméra $C$ sont illustrés en figure A.1, le septuplet de description correspondant est :

$$C = (X_C, Y_C, Z_C, \phi_C, \theta_C, \psi_C, \gamma_C)$$

Une autre représentation possible et classiquement utilisée, consiste à modéliser la caméra par deux vecteurs et une distance focale :

- le premier vecteur correspond à la position 3D de la caméra,
- le deuxième est utilisé pour représenter l'orientation, *i.e.* le vecteur *look-at* de la caméra.

Les deux représentations sont équivalentes à condition que le vecteur *look-at* soit normalisé. Dans ce cas, il est possible de passer indifféremment de l'une à l'autre des représentations, c'est-à-dire d'utiliser soit la représentation basée sur les angles d'Euler, soit la représentation vectorielle.

### A.0.2 Modèle de représentation d'un objet 3D

Les objets 3D peuvent être mobiles ou non suivant que nous les étudions en environnements dynamiques (cf. chapitre 5) ou statiques (cf. chapitres 3 et 4). Nous donnons maintenant la représentation d'un objet dynamique, cette dernière étant plus générale, dans les cas statiques certains paramètres ne seront donc pas pris en compte.

Un objet 3D est illustré en figure A.2 et est caractérisé par :

- une position 3D,
- une orientation intrinsèque (un vecteur 3D) représentant l'orientation générale de l'objet. Cette dernière correspond par exemple à la direction du regard d'un personnage.

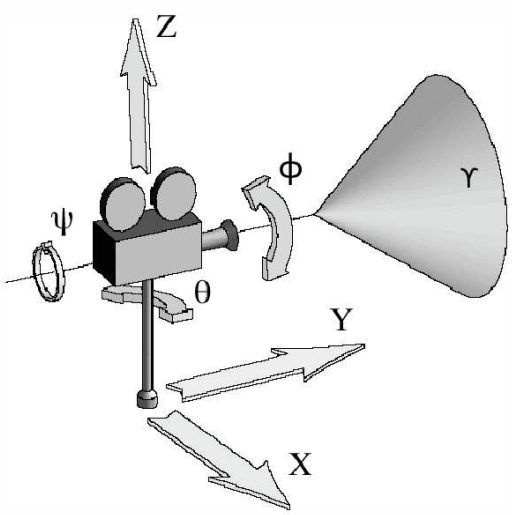



Figure A.1 – Modèle de représentation d'une caméra basée sur les angles d'Euler.

- un rayon définissant une sphère englobante. Le choix d'une sphère comme englobant est criticable, en effet, il est particulièrement mal adapté pour des objets filiformes. Toutefois, nous avions besoin de ce type d'englobant pour la réalisation de notre méthode des *volumes sémantiques* présentée au chapitre 4.
- une trajectoire permettant de définir ses mouvements au cours du temps,
- un vecteur vitesse caractérisant sa vitesse de déplacement le long de sa trajectoire.

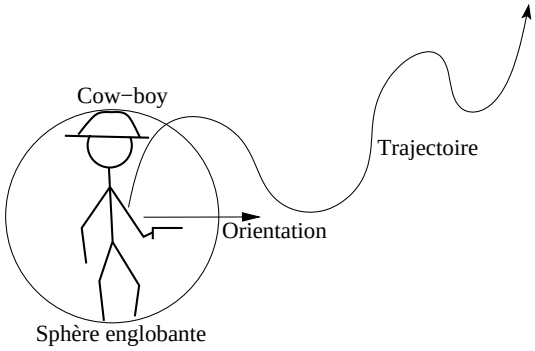



Figure A.2 – Modèle de représentation d'un objet 3D.

# ANNEXE B

# Le logiciel ZDM

Cette annexe est consacrée au logiciel ZDM qui nous a servi d'outil de démonstrations pour la méthode des *volumes sémantiques* présentée au chapitre 4. Ce logiciel a été développé au sein de la thèse de M. Christie [Chr03] par Dominic Christie et lui même, dans le cadre d'un projet industriel avec la société MédiaForce et financé par les projets multimédia PRIAMM du CNC (Centre National de la Cinématographie). La réalisation a nécessité sept mois de développement et environ 40000 lignes de code. Ce logiciel est un modeleur tridimentionnel dont l'objectif principal est d'assister l'utilisateur dans la création de mondes virtuels. La figure B.1 présente une capture d'écran de l'outil.

Voici les principales fonctionnalités de ce modeleur :

- création de cartes $3D$ à partir de cartes de niveaux,
- création de tunnels en se basant sur des techniques de surfaces implicites et de cylindres généralisés,
- augmentation/diminution du maillage (niveau de détail),
- placement d'objets à la volée sous contraintes,
- architecture modèle/instance,
- manipulation hiérarchie/objets/facettes/points,
- blabla

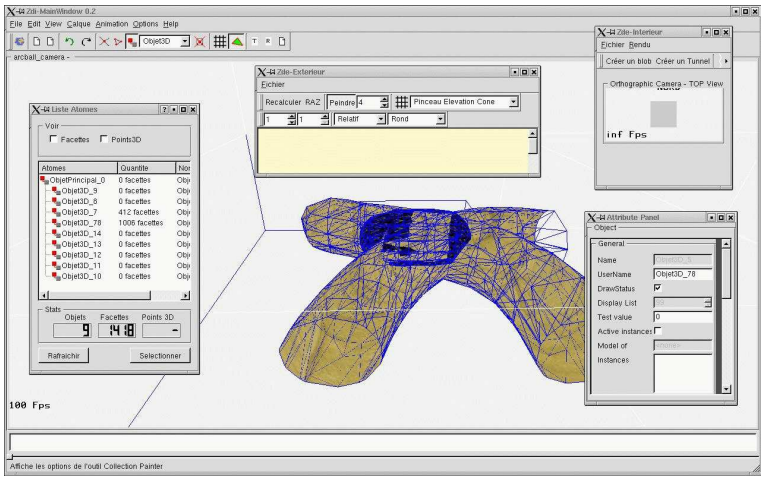



Figure B.1 – L'outil de modélisation ZDM.

Nous utilisons cet outil dans le but de visualiser les volumes sémantiques crées et de pouvoir visualiser les différents résultats obtenus correspondants à chacune des classes de solution d'un problème

de placement de caméra. La communication entre l'implémentation des volumes sémantiques et ZDM est effectuée *via* un *plug-in*. Ce dernier permet de modéliser les volumes sémantiques en tant qu'entités affichables par ZDM et de créer une caméra correspondant à chacun des meilleurs candidats de chaque classe de solution. Nous avons également implémenté une fonction permettant de changer interactivement de caméra afin de permettre à l'utilisateur de naviguer entre les différents volumes sémantiques et de se rendre compte des caractéristiques propres à chacune des caméras correspondantes.

# ANNEXE C

# Implémentation du cadre de recherche locale continue

Nous présentons maintenant brièvement l'implémentation de notre extension continue du cadre de la recherche locale continue (RLC) ainsi qu'une application de la métaheuristique taboue aux domaines définis par des intervalles à bornes flottantes.

Nous avons utilisé la bibliothèque ELISA dédiée à la résolution de problèmes de satisfaction de contraintes numériques elle même basée sur la bibliothèque GAOL de gestion de l'arithmétique des intervalles. Cette bibliothèque est développée au sein de l'équipe XXX du laboratoire LINA de l'Université de Nantes. Le diagramme de classe UML présentant cadre de l'extension continue de la recherche locale avec implémentation d'une métaheuristique taboue est illustré en figure C.1. Le développement de cette extension a été fait en C++ afin de tirer partie de la flexibilité obtenue par l'utilisation des patrons de classe (*templates*) de ce langage.

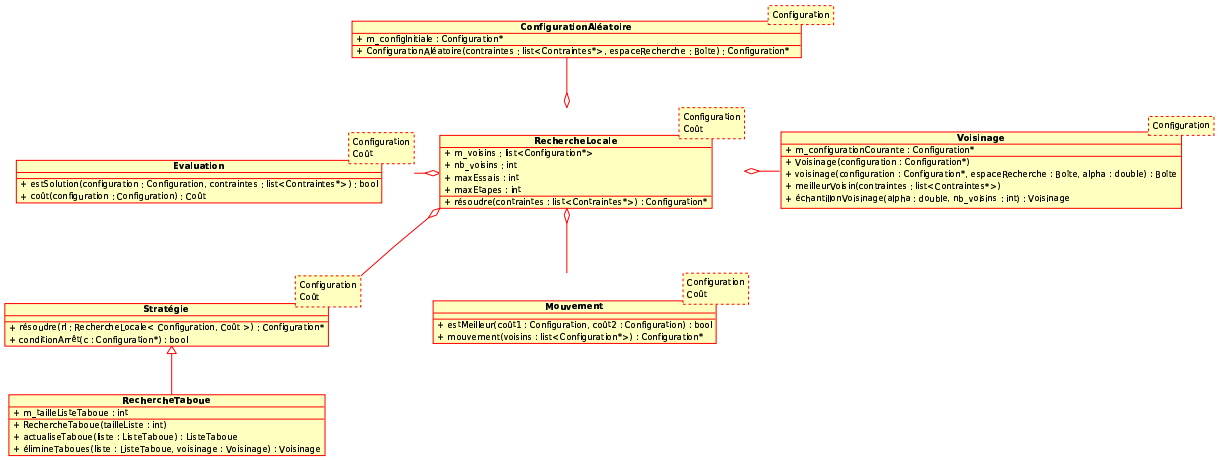



Figure C.1 – Diagramme de classes de l'implémentation de l'extension de la recherche locale avec recherche taboue aux domaines continus.

Afin d'assurer la généricité de notre cadre continu de recherche locale, nous avons choisi de configurer les classes du cadre RLC par deux classes : une définissant les configurations continues et une concernant le coût d'une de ces configurations par rapport à la satisfaction des contraintes. La classe

RechercheLocale représentant la cadre continu de recherche locale (RLC) consiste en une agrégation des classes Mouvement, Voisinage, Evaluation, ConfigurationAléatoire et Stratégie. Les quatres premières classes correspondent directement aux éléments clés d'une recherche locale. Un cinquième composant, Stratégie, est ajouté afin de permettre la configuration de la manière d'agencer ces quatre composants élémentaires.

La classe ConfigurationAléatoire implémente la génération aléatoire d'une configuration dans un espace de recherche, cette classe est également utilisée lors des réinitialisations aléatoires (*random restarts*) de la phase de diversification de l'algorithme classique de recherche locale. La classe Voisinage, comme nous l'avons présenté en section 3.2.13.2, permet de générer un ensemble de voisins par rapport à une configuration courante, un espace de recherche et un facteur de réduction (le réel alpha). Le composant Mouvement va quant à lui déterminer les futures configurations courantes de l'algorithme en se basant sur la comparaison de l'évaluation de deux configurations fournie par la classe Evaluation utilisée dans la fonction Mouvement::estMeilleur($coût_1$, $coût_2$).

La classe Stratégie permet de combiner les pas de générations de voisins, d'évaluations et de mouvements, afin d'assurer la généricité de l'implémentation proposée. Nous fournissons par ailleurs une classe dérivée RechercheTaboue implémentant la métaheuristique taboue telle que présentée dans la section 3.2.13.5.

# ANNEXE D

# Le logiciel USV : un outil de visualisation de pavés

L'objectif principal de l'outil USV (*Universal Solution Viewer* [Chr00, Chr03]) est de visualiser des données. Dans la version actuelle, les données sont des pavés dont chaque dimension est un intervalle de réels (*c.f.* arithmétique des intervalles sur les flottants). La méthode de représentation de ces données passe par l'écriture de scripts spécifiques. La figure D.1 présente une évaluation sur les intervalles de la fonction $f(z) = \sin x \cos y$.

Dans une première partie, nous présentons notre outil : quels sont les besoins et quelles en sont les fonctionnalités ? Ensuite, le format du fichier d'entrée est introduit, suivi de l'écriture des scripts de représentation. Enfin, nous présentons l'extension de l'outil par la création de modules.

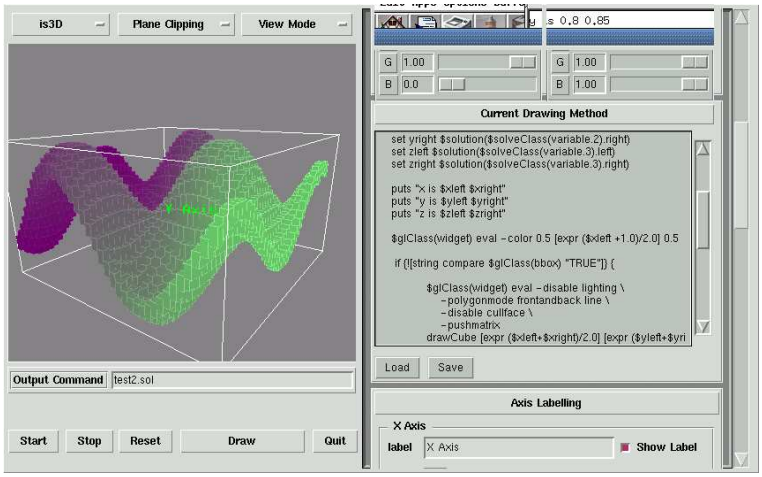



Figure D.1 – Visualisation de la fonction $f(z) = \sin x \cos y$ sur les intervalles avec l'outil USV.

### *D.0.2.1 Format du fichier d'entrée*

Le fichier d'entrée est composé de deux parties : la définition des variables et la liste des pavés.

**Définition des variables** Cette partie est optionnelle : si elle n'est pas présente, l'outil parcourt la liste des pavés et met à jour les bornes gauche et droite du domaine de chaque variable. Sinon, il est possible de définir les variables : à chacune doit être associée la description suivante :

```
Variable <var1> with domain <valeur>
```

Le séparateur de tokens est le caractère espace : si une variable doit contenir un espace, englobez celle-ci dans des double quotes, *e.g.* "ma variable". Il est évident que chaque variable possède un nom unique (aucun test n'est fait dans le cas contraire).

**Exemple D.10 (Définition des domaines).**

```
Variable x with domain [ 0, 5 ]
Variable y with domain [ -5, 5 ]
```

**Liste des pavés.** Chaque "entrée" possède le format suivant :

```
SOLUTION { { <var1>  <valeurs> } { <var2>  <valeurs> } ... }
```

var*i* est le nom d'une variable et *valeur* représente son domaine, soit ici un intervalle de réels.

**Exemple D.11 (Liste des pavés).**

```
SOLUTION { { x -4.0625 -3.75 }  { y -2.62496 -2.5 }  { t 0 3 } }
SOLUTION { { x -2.8125 -2.5 }  { y -2.81346 -2.5 }  { t 0 3 } }
```

### *D.0.2.2 Le script de représentation graphique*

**Exemple** Soit un ensemble de données à trois variables $(x, y, z)$. Voici un script de représentation graphique possible :

```
proc {genericDrawSolution.dummy} {} {

  # set color to red
  setColor 1 0 0

  # draw a cube
  drawCube [getIntervalMiddle x] [getIntervalMiddle y] [getIntervalMiddle z] \
           [getIntervalWidth x] [getIntervalWidth y] [getIntervalWidth z]
}
```

### *D.0.2.3 Extension par modules*

Une des motivations principales à la création de cet outil est de pouvoir facilement l'étendre à l'aide de modules (ou de composants, c'est plus actuel).

# Placement de caméra en environnements virtuels

Jean-Marie NORMAND


## Résumé

Le placement de caméra en environnement virtuel consiste à positionner et orienter une caméra virtuelle 3D de façon à respecter un ensemble de propriétés visuelles ou cinématographiques définies par l'utilisateur. Réaliser cette tâche est difficile en pratique. En effet, l'utilisateur possède une vision claire du résultat qu'il souhaite obtenir en termes d'agencement des objets à l'image. Toutefois le processus classique de placement de caméra est particulièrement contre-intuitif. L'utilisateur doit effectuer une inversion mentale afin d'inférer la position et l'orientation de la caméra dans l'environnement 3D amenant au résultat souhaité. Des méthodes d'aide au placement de caméra apparaissent donc particulièrement profitables pour les utilisateurs. Dans cette thèse, nous identifions trois axes de recherche relativement peu couverts par la littérature dédiée au placement de caméra et qui nous apparaissent pourtant essentiels. D'une part, les approches existantes n'offrent que peu de flexibilité tant dans la résolution que dans la description d'un problème en termes de propriétés visuelles, en particulier lorsque celui-ci ne possède aucune solution. Nous proposons une méthode de résolution flexible qui calcule l'ensemble des solutions, maximisant la satisfaction des propriétés du problème, que celui-ci soit sur-contraint ou non. D'autre part, les méthodes existantes ne calculent qu'une seule solution, même lorsque le problème possède plusieurs classes de solutions équivalentes en termes de satisfaction de propriétés. Nous introduisons la méthode des volumes sémantiques qui calcule l'ensemble des classes de solutions sémantiquement équivalentes et propose un représentant de chacune d'elles à l'utilisateur. Enfin, le problème de l'occlusion, bien qu'essentiel dans la transmission de l'information, n'est que peu abordé par la communauté. En conséquence, nous présentons une nouvelle méthode de prise en compte de l'occlusion dans des environnements dynamiques temps réel.


**Mots-clés :** placement de caméra, CSP numériques, approximation intérieure, occlusion.


## Abstract

Virtual camera composition consists in positioning and orienting a camera within a 3D environment in order to fulfil a set of user-defined visual or cinematographic properties. This task is difficult to achieve in practice. Indeed, while the user knows exactly how the objects should be laid out in the resulting image, the camera's 3D software manipulation process is awkward. The user has to perform a mental inversion i.e., while knowing the result, he has to infer the camera's position and orientation in the 3D environment which will lead to the correct layout on the screen. As a result, the user would greatly profit from the development of virtual composition assisting techniques. In this thesis, we have identified three research interests related to virtual camera composition that deserve specific attention. Firstly, existing camera control techniques lack flexibility both in the resolution and the description processes. Over-constrained problems (i.e. without any solutions) are but not supported and describing a camera composition problem by a set of visual properties can be difficult. We propose a flexible resolution technique that computes the solutions by maximizing the number of their properties, regardless of whether the problem is over-constrained or not. Secondly, existing methods only compute a unique solution for a virtual camera composition problem, even when the problem possesses a set of equivalent solutions with respect to the properties. We introduced the semantic volumes method which computes every class of semantically equivalent solution and presents the user with an appropriate representative for each class. Finally, occlusion consists of a fundamental problem regarding the conveyance of information contained in images. Consequently, we have proposed a new method for occlusion avoidance in dynamic real-time virtual environments.


**Keywords:** camera control, numeric CSPs, inner approximation, occlusion.

### Classification ACM

Catégories et descripteurs de sujets : D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*; G.1.0 [**Numerical Analysis**]: General—*Interval arithmetic*; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Animations*

Termes généraux : Algorithms, Theory, Reliability